经典计算机科学著作最新修订版

计算机程序设计艺术 第1卷 基本算法

(第3版)

国防工业出版社
National Defense Industry Press
http://www.ndip.cn

Addison Wesley

经典计算机科学著作最新修订版

计算机程序设计艺术
第1卷 基本算法
(第3版)

The Art of Computer Programming

苏运霖 译

(美) DONALD E. KNUTH 著

Addison Wesley

国防工业出版社
National Defense Industry Press
http://www.ndip.cn

第 1 卷 / 基本算法（第 3 版）

丛书第 1 卷以基本的程序设计概念和技术开始，然后专注于信息结构——计算机内部信息的表示、数据元素之间的结构关系及其有效处理方法。描述了模拟、数值方法、符号计算、软件与系统设计的初等应用。新版本增加了几十项简单但重要的算法和技术，并对有关数学预备知识做了大量修正以适应现时研究的趋向。

第 2 卷 / 半数值算法（第 3 版）

第 2 卷对半数值算法领域做了全面介绍，分“随机数”和“算术”两章。本卷总结了主要算法范例及这些算法的基本理论，广泛剖析了计算机程序设计与数值分析间的相互联系。第 3 版中特别值得注意的是 Knuth 对随机数生成程序的重新处理和对形式幂级数计算的讨论。

第 3 卷 / 排序和查找（第 2 版）

第 3 卷的头一次修订对经典计算机排序和查找技术做了最全面的考察。它扩充了第 1 卷对数据结构的处理，以将大小数据库和内外存储器一并考虑；遴选了精心核验的计算机方法，并对其效率做了定量分析。第 3 卷的突出特点是对“最优排序”一节的修订和对排列论与通用散列法的讨论。

经典计算机科学著作最新修订版

计算机程序设计艺术
第1卷　基本算法
（第3版）

The Art of Computer Programming

苏运霖　译

（美）DONALD E. KNUTH 著

Addison Wesley
国防工业出版社
National Defense Industry Press
http://www.ndip.cn

经典计算机科学著作最新修订版

计算机程序设计艺术

第1卷

基本算法

（第3版）

[美] Donald E. Knuth 著
苏运霖 译

国防工业出版社
·北京·

著作权合同登记号　图字:军-2001-018号

图书在版编目(CIP)数据

计算机程序设计艺术.第1卷,基本算法:第3版/(美)克努特(Knuth,D.E.)著;苏运霖译.—北京:国防工业出版社,2007.6重印

书名原文:The Art of Computer Programming

ISBN 978-7-118-02799-0

Ⅰ.计…　Ⅱ.①克…②苏…　Ⅲ.电子计算机-算法设计　Ⅳ.TP311

中国版本图书馆CIP数据核字(2002)第007597号

Simplified Chinese edition copyright © 2002 by Pearson Education Inc. and National Defense Industry Press.
Original English language title: The Art of Computer Programming, Vol.1, Foundamental Algorithms, 3rd Edition by Donald E. Knuth
Copyright © 1997 by Addison Wesley Longman
All Rights Reserved.

互联网网页 http://www-cs-faculty.stanford.edu/~knuth/taocp.html 包含本书及相关著作的最新信息。

※

国防工业出版社出版发行

(北京市海淀区紫竹院南路23号　邮政编码100044)

北京奥鑫印刷厂印刷

新华书店经售

*

开本 710×960　1/16　**印张** 40½　**字数** 880千字

2007年6月第3次印刷　**印数** 7001—10000册　**定价** 98.00元

(本书如有印装错误,我社负责调换)

国防书店:(010)68428422　发行邮购:(010)68414474

发行传真:(010)68411535　发行业务:(010)68472764

中文版前言(附原文)

谨向拥有此书的全体中国读者问候！我诚挚地向中国的程序员推荐我的中文名字"高德纳"，那是 1977 年在我访问你们的国家之前，由姚储枫(Francis Yao)为我起的。那次为期三周的访问至今仍给我留下美好的回忆。我高兴地看到自 1989 年以来，在《计算机科学与技术》杂志的刊头上有"高德纳"的名字。虽然我不通晓你们的语言，但这个名字已使我对全体中国人民倍感亲近。

献身于计算机程序设计的人们必须从事大量艰苦的工作，必须掌握许多精微的技术细节，因而在这方面深有造诣者不多，但其回报却是巨大的，因为一个编写得很好的程序完全可以成为一件美丽的艺术品，而且还因为，正是计算机程序使得全世界的人们彼此靠得更近。

Donald E. Knuth (高德纳)

Preface to the Chinese Edition

Greetings to all readers of these books in China! I fondly hope that many Chinese computer programmers will learn to recognize my Chinese name "高德纳", which was given to me by Francis Yao just before I visited your country in 1977. I still have very fond memories of that three-week visit, and I have been glad to see "高德纳" on the masthead of the *Journal of Computer Science and Technology* since 1989. This name makes me feel close to all Chinese people although I cannot speak your language.

People who devote much of their lives to computer programming must do a great deal of hard work and must master many subtle technical details. Not many are able to do this well. But the rewards are great, because a well written program can be a beautiful work of art, and because computer programs are helping to bring all people of the world closer together.

Donald E. Knuth (高德纳)

译 者 前 言

在新千年到来的前夕，我在译稿上写下最后一个句号。这意味着几百万字的重译工作至此告一段落。我顿觉如释重负，浑身轻松；却又思潮澎湃，心绪激动。端笔记下，遂成此前言，留给己与人。

二十多年前，当中国大地还笼罩在"知识分子臭老九"的气氛下时，我在书店中偶然发现 D.E. 克努特(高德纳)这套传世之作，立即预感到它的巨大精神力量，遂萌生把它们译出，以供我国同仁共同从中获益的想法。当我把这个想法讲给当时和我一起共事、曾经是我老师的管纪文先生时，他也欣然同意，并愿意和我一起完成这件既苦又累、但又是很有意义的工作。

当年，当我和管纪文先生完成了全书的翻译工作时，我也曾有过这样一种兴奋激动的心情，今天，我似乎是在重新经历那时的感觉。

也许每一个曾经经历过那个时期的人们都会为二十多年来沧海桑田般的变化而感慨万千。当年由于众所周知的原因和社会气氛，为了出版这套书，实在是颇费周折的。因此我们要在这里感谢当年为使我们的译著得以出版的许多领导和专家，正是他们的真知灼见和无私无畏的精神，以及他们不畏艰难的支持和帮助，才使我们的工作免于"胎死腹中"。二十多年来的改革开放，确实为我们国家、为我们的人民带来了巨大变化。正是在这样的大气候下，才有我们这项工作的继续和这套书在中国的再版。

我深深地为作者锲而不舍、精益求精的精神所感动。作者在这套书第 1 版的前言中，曾经提到了他的十分复杂的感情——他说，从一方面说，他当然希望自己的书能永远地有价值，永远给人们提供教益；但另一方面，他也知道，对于迅猛发展的像计算机这样的学科，知识的更新是如此迅速，要想使自己的书经久不衰，永不过时，是做不到的。所以他只想使自己的书不至于或不要过时太快。然而，十分可贵的是，他没有停留在这种矛盾中，而是以一种不屈不挠的精神来延续自己著作的寿命和价值。为了做到这一点，他花费了大量的精力，投入了巨大的劳动对全书进行修订；最重要的是以当前这个领域的知识前沿来全面更新全书。他还不满足于此，他还要把书中原来使用的样板机 MIX 及其语言，更新成为新一代 MMIX。在作者身上，精益求精的精神体现得十分生动：对于他所研制的 TEX 和 METAFONT 是这样——可以说他是以十年铸一剑来完成这两个誉满全球的计算机排版印刷的精品的；对于他的科学研究是这样——可以说他总是要把每个所研究的问题都彻底解决了才甘罢休的；对于他的这套成名著作也是如此——他不但要使其内容经得起考验，也要让全书绝无错误。因此现在他又承诺以 2.56 美元的奖励来酬谢任何一个错误的头一个发现者。这种对工作高度认真负责，一丝不苟的精神，难道不值得我们为之赞叹，不值得我们立为楷模吗？

二十多年过去了,我和管纪文先生共同翻译的第1版离开书店已经很久了。在一些图书馆里,即使能找到,大概也早已皮破纸黄了。然而每当人们谈起它们时,仍然会以赞扬的口吻说,那几本书翻译得很好,对我国计算机的发展做出了很大贡献。我要感谢读者和同行们的鼓励。今天,我所期盼的仍然是,不仅得到读者和专家们对我付出的艰苦劳动的肯定,而且更希望我付出的这一劳动,对于我国的计算机界——特别是年轻一代的计算机界,产生积极的影响。我盼望他们认认真真地去阅读这套书(而且还不是为了作为找比尔·盖茨的敲门砖,尽管他说过,任何一个读懂这套书的人,都将得到他的赏识——他可以接受你的简历),以便为献身于我国自己的计算机事业打下基础;我期望他们从这套书学到的远不仅仅是作者传授的知识,还要学到他严谨的治学态度、孜孜以求的探索精神和创新意识,更期盼他们以此为基础去创新、去攀登知识的巅峰。

作者计划,这套书共有7卷,但迄今他真正完成的还只有这头3卷。不过他已在全力以赴赶写第4和第5两卷,希望在两三年内使它们问世。最后,再争取把第6卷和第7卷全部完成。我期盼的是能够看到这全套书的大功告成,而我自己也能够把它们全部译完,奉献给我国的读者,这也算是我对我国计算机事业所做的一点点贡献吧!

顺便提到,本书得以问世,和许多人的帮助是分不开的。在这里,谨向为本书付出了劳动的江门五邑大学的孟亚老师,我的研究生杜民、胡茂伟以及暨南大学计算机科学系96级和97级的张永源、吕双欢、卢国旺等同学表示感谢。

并以此书献给已故中国科学院学部委员、我国计算机事业的先驱者之一、为我国计算机事业的发展做出重大贡献的我的老师王湘浩教授。

前　　言

Here is your book, the one your thousands of letters have asked us
to publish. It has taken us years to do, checking and rechecking countless
recipes to bring you only the best, only the interesting, only the perfect.
Now we can say, without a shadow of a doubt, that every single one of them,
if you follow the directions to the letter, will work for you exactly as well
as it did for us, even if you have never cooked before.
这就是你们要的书，是你们数不清的来信要求我们出版的书，
它已花了我们多年的时光，我们一遍又一遍地咀嚼每一个细节，检查每一道食谱，
惟一的愿望是把最好的、最有趣的和最完美的菜肴奉献给你们。
现在我们可以毫无疑虑地说，
如果你精心仿效，即使你以前从未烹饪过，
本书任何一道食谱都将如它为我们提供的那样，也为你提供最精致美味的食物。
—*McCall's Cookbook* (1963)

为数字计算机编写程序的过程是饶有趣味的，因为它不但具有经济和科学价值，也是犹如赋诗或作曲那样的美学实践。本书是多卷集的第 1 卷，旨在对读者进行作为熟练的程序员所必需的各种技巧的训练。

以下各章并不打算作为计算机程序设计的入门介绍。我们假定在此之前，读者已经有某些经验。先决条件实际上很简单，但初学者需要时间和实践才能理解数字计算机的概念。读者应具有：

a）一台存储程序式数字计算机如何工作的概念；但不必是电子学方面的，而是指令如何被保存于机器存储器中以及逐个地被执行的方式。

b）把问题的解法翻译成为计算机能“理解”的明确术语的能力。（这些机器没有普通感觉，它们只能按照告知它们的要求不折不扣地去做。当人们开始试图使用计算机时这是最难以掌握的概念。）

c）最基本的计算机技术的某些知识，例如循环（重复地执行一组指令），使用子程序以及使用下标变量的某些知识。

d）普通计算机术语——“存储器”、“寄存器”、“位”、“浮点”、“溢出”、“软件”等方面的少量知识。在正文中未予定义的大多数词，在每卷末尾的索引中都给出简明的定义。

这四个先决条件也许可以概括成一个要求，即读者至少应该为一台计算机编写并调试过至少四个程序。

我力图使这一丛书满足多方面的需要。首先，这些书是一套参考书，它汇总了在若干重要领域的知识。其次，它们可以作为计算机科学和信息科学领域的大学教材或自学读物。为了实现这两方面的目标，我在本书中加入了大量的习题并为大多数习题附上了答案。我还尽力用实例充实本书的内容，而避免作含糊空泛的评论。

这套丛书的对象，不是对计算机仅仅有一时兴趣的那些人们。但这绝不是说，这部书仅仅是为计算机的专业人员写的。事实上，我的主要目标之一是使这些程序设计技术可以被工作在其它领域的许多人掌握；这些人能够有效地使用计算机，却没有时间搜寻隐藏在各种技术杂志中的必需信息。

我们可以把这些书的主题称做“非数值分析”。计算机在传统上是同求解数值问题，例如一个方程的根，插值与积分等等的求解与计算相关联的。但在这里，我们并不处理这样一些课题，除非是顺便提及。数值计算机程序设计是极为有趣和迅速发展的领域，因而已有许多这方面的书出版。然而，自 20 世纪 60 年代初期开始，计算机已经更频繁地用于数值仅是偶尔出现的一些问题上；这里所使用的是计算机的判定能力，而不是它的算术运算能力。在非数值问题中，我们也用一些加法和减法，但我们很少需要乘法和除法。当然，即使是主要对数值计算机程序设计感兴趣的人也将从非数值技术的学习中获益，因为非数值技术在数值程序中同样存在。

非数值分析的许多研究成果分散于许多技术杂志中。我的方法是通过研究那些最基本的技术，来提取大量文献的精华。所谓最基本是指它们可被应用于许多类型的程序设计场合。我尽力把这些思想上升为“理论”，并解释如何把这一理论应用于广泛的实际问题中。

当然，“非数值分析”是关于这个研究领域的极其负面 * 的名称，最好应该有一个正面的描述性的术语来表征这个课题。“信息处理”对于这个内容而言太宽了，而“程序设计技术”又太窄了。因此，我希望以*算法分析*作为这套书所涵盖课题的恰当名称，这一名称隐含着“特定计算机算法性质的理论”的意思。

整套丛书，以“计算机程序设计艺术”为书名，其总目录如下：

第 1 卷　基本算法
　　第 1 章　基本概念
　　第 2 章　信息结构
第 2 卷　半数值算法
　　第 3 章　随机数
　　第 4 章　算术
第 3 卷　排序与查找
　　第 5 章　排序
　　第 6 章　查找
第 4 卷　组合算法
　　第 7 章　组合检索

---

* 因为用的是在“数值”之前加上“非”这样一种表述。——译者注

第 8 章　递归

第 5 卷　语法算法

第 9 章　词法扫描

第 10 章　语法分析

第 4 卷所处理的课题相当之大,它实际上包含 3 个分册(卷 4A, 4B 和 4C)。有关更专门课题的另外两卷也在准备中:第 6 卷语言理论(第 11 章)和第 7 卷编译程序(第 12 章)。

我从 1962 年开始按照这个章次来写一本书。但我很快发现,更重要的是要深入地讨论这些课题而不是浅尝辄止。我写出的文稿的篇幅使我意识到,每章所含的内容都已足以作为一学期的大学课程,因此把这部书分成几卷出版是明智的。我知道,在一本书中仅写一两章是很少见的,但我决定仍保留原来的章目以便于交叉参考。我也计划出一本第 1 卷到第 5 卷的缩写本,专门用作本科计算机课程的更通用的参考书或教材,其内容是这些书内容的一个子集,略去了其中更为专门的信息。在缩写本中的章目仍和全集中的一样。

本卷可看做整个丛书的一个"交叉点",意思是,它包含了在所有其它书中用到的基本材料。另一方面,第 2 卷至第 5 卷可以彼此独立地阅读。第 1 卷不仅可作为同其它卷相关联的一本书,而且还可以作为*数据结构*(重点是第 2 章的内容)、*离散数学*(重点是 1.1,1.2,1.3.3 和 2.3.4 等节的内容)或者*机器语言程序设计*(重点是 1.3 和 1.4 节的内容)等课题的大学教材或自学教材。

在写这些章节时,我的宗旨不同于当代许多有关计算机程序设计的书,我并不试图教读者如何使用某些人的软件,我所关心的是教人们写更好的软件本身。

我原来的目标是把读者带到所讨论的每一课题的知识的前沿。然而,要在经济上可以盈利的一个领域保持不落伍是极其困难的,而且计算机科学的迅速发展使这一梦想已不可能实现。这一课题已经变成一个巨大的花毯,它由来自全世界的数以万计的智者所奉献的数以万计的精妙结果织成。因此我的新的目标已变成专注于那些"经典"的技术(它们可能在今后数十年间仍保持其重要性),并尽我之所能来描述它们。特别是,我已试图来跟踪每一课题的历史,并且为未来进展提供一个坚实基础;我尝试选择简明并同当前用法一致的术语;我试图把既优美而又易于表述的有关顺序计算机程序设计的所有已知的思想都包括进来。

关于本丛书的数学内容还有几句话要说一说。我已把这些材料组织成,使得只需有高中代数知识的人就可读懂它,他们只须跳过更深入的数学部分即可。同时,那些数学功底好的读者将能学习同离散数学有关的许多有趣的数学技巧。为了兼顾这两方面的需要,我对每个习题都指定级别,并把那些主要是数学的习题特地标了出来,同时,在大多数章节中,把主要的数学结果放在它们的证明之前来叙述。这些证明或者被留作习题(其答案可在一个独立的部分找到),或者在同一节的末尾中给出。

主要是对程序设计而不对相关的数学感兴趣的读者可以在一旦对数学内容感觉到困难时就停止阅读它的大部分内容。但另一方面,面向数学的读者将在这里发现丰富、

有趣的内容。有关计算机程序设计的许多发表过的数学知识都是有错误的。因此本书的目的之一是向读者讲授有关这一课题的适当的数学内容。由于我自认为是一名数学家,因此尽我所能来维护数学的完整性是我的义务。

对于本丛书中的大多数数学内容而言,有初等微积分的知识就足够了。因为需要的大多数其它理论都是由此而建立起来的。然而,有时我确实需要使用复变函数论、概率论、数论等等较深入的定理。在这种情况下,我会列出讨论这些课题的适当教材供读者参考。

在编写这些书时我要做出的最难的决策在于介绍这些不同技术的方式,流程图和算法的逐步非正规描述的优点是众所周知的。关于这方面的讨论,见论文“Computer Drawn Flowcharts”,ACM Communications,第6卷(1963年9月),555~563。同时,一种正规的精确语言对于描述任何计算机算法也是必要的,因此我需要决定,为此目的是使用一种代数语言例如 ALGOL 或 FORTRAN 呢,还是使用面向机器的语言。或许许多当今的计算机专家都不同意我使用一种面向机器的语言,然而我已深信,由于下列原因,它肯定是一个正确的选择:

a) 程序员在很大程度上受他写程序时所用的语言的影响;有一个压倒性的趋势,就是宁愿使用在语言中最简单的结构,而不会去使用对于机器而言最好的那些东西。通过理解一种面向机器的语言,程序员就将趋向于使用有效得多的方法;这更接近于现实。

b) 除去少数例外,我们所要求的程序都是比较短的,因此只要有一台适当的计算机,理解这些程序将毫无麻烦。

c) 对于讨论诸如共行程序(coroutine)链接、随机数生成、多精度算术以及涉及存储器的有效使用等许多问题的重要的低级细节而言,高级语言是不适当的。

d) 一个真正对计算机感兴趣的人,应当很好地学习机器语言,因为它是一台计算机的基本部分。

e) 无论如何,作为在许多例子中叙述的软件程序的输出,某种机器语言将是需要的。

f) 新的代数语言每5年就走红和过时,而我想强调的是永恒的概念。

我承认,从其它观点来看,以高级语言来写程序是稍微容易一些的,而且调试程序要更容易。确实,自1970年以来,对于我自己的程序,我也很少使用低级的机器语言,因为计算机是如此之快和如此之强了。然而对于本书中我们感兴趣的许多问题而言,程序员的技巧是至为重要的。例如,某些组合计算需要被重复1万亿次。因此如果我们能从它们的内循环挤出1微秒来,就可以节约出11.6天来。类似地,对于在许多计算机装置中每天要被用上许多次的软件编写来说,作出额外的努力是值得的,因为这个软件只须编写一次。

在做出了使用面向机器的语言的决策之后,该使用哪一种语言呢?我可以选择一个特定的机器 X 的语言,但如果那样,不具有机器 X 的人们就会认为本书是仅为使用 X 的人们写的。而且,机器 X 可能有大量的特性,它们同本书的内容全然无关,但我们

仍须加以说明。况且在两年内，机器 X 的厂家将推出机器 $X+1$ 或 $10X$ 来，那机器 X 将不再使任何人感兴趣了。

为避免陷入这种窘境，我试图设计一种具有非常简单的操作规则（比如说，只要花上一小时便可学会）的"理想"计算机，而它又非常类似于真实的机器。学生毫无理由害怕领会一种以上的计算机的特性；一旦掌握了一种机器语言，其它语言就很容易吸收了。确实，认真的程序员可以预期，在他们的事业中会遇到许多不同的机器语言。因此，一台虚构的机器剩下的惟一缺点是执行为它编写的任何程序的困难。幸而，这实际上不是一个问题。因为许多志愿者已经站出来为这假想的机器写了模拟程序。这样的模拟程序对于教学的目的而言是理想的，因为它们比起一台实际的计算机来更易于使用。

我试图在每个主题中引用早期的最好论文及最新的作品。当引用参考文献时，我都使用这些期刊的标准缩写，但以下最普遍被引用的杂志例外，它们的缩写如下*：

CACM = Communications of the Association for Computing Machinery（《美国计算机协会通讯》）

JACM = Journal of the Association for Computing Machinery（《美国计算机协会杂志》）

COMP. J. = The Computer Journal (British Computer Society)（《计算机杂志》——英国计算机学会）

Math. Comp. = Mathematics of Computation（《计算数学》）

AMM = American Mathematical Monthly（《美国数学月刊》）

SICOMP = SIAM Journal of Computing（《SIAM 计算杂志》）

FOCS = IEEE Symposium on Foundations of Computer Science（《IEEE 计算机科学基础论文集》）

SODA = ACM-SIAM Symposium on Discrete Algorithms（《ACM-SIAM 离散算法论文集》）

STOC = ACM Symposium on Theory of Computing（《ACM 计算理论论文集》）

Crelle = Journal für die reine und angewandte Mathematik （《纯粹和应用数学杂志》）

例如，"*CACM* **6** (1963), 555 ~ 563"表示上页中提到的参考文献；我也使用*CMath* 来代表*Concrete Mathematics* 这本书，它在 1.2 的引言中被提到。

这一套书中的许多技术内容还出现于习题中。当一道较难的习题背后的思想不属于我自己的时候，我尽量提及提出该思想的人来。对于相应的参考文献的引用通常则在该节伴随的文字中给出，或者在习题的答案中给出，但在多数的情况下，习题都基于未公开发表过的材料，对于它们就无法进一步给出参考文献了。

在编写这些书的过程中，我获得了许多人的帮助，对此我深为感激。首先我要感谢我的妻子 Jill，感谢她无限的耐心，感谢她为本书绘图，感谢她数不尽的各种帮助。其

* 鉴于参考文献直接引用原文更便于读者检索，本书保留参考文献原有文字而不译出。读者可从书末索引与词汇表中查到文献和人名的译名，或从所附网址查找有关信息。对于本书涉及的人名，除极少数众所周知者外，均直接使用原文而未译出。——本书责任编辑

次，我要感谢 Robert W. Floyd，他自 20 世纪 60 年代以来为扩充本书的内容奉献了大量的时间。还有上千人也提供了重要的帮助——光列出他们的名字就将是一本书了。他们中的许多人善意地允许我使用他们迄今为止未发表的作品。我在 Caltech(加州理工学院)和 Stanford(斯坦福)的研究有许多年都得到国家科学基金和海军研究办公室的慷慨支持。自 1962 年我开始本项目以来，Addison-Wesley 一直提供了极好的帮助与合作。我知道向每一个人表示感谢的最好方法是通过这一出版物展示出，他们的投入成就了这些书，我想，这些书正是他们要我写成的那个样子。

在花费了 10 年时间研制用于计算机排版的 TEX 和 METAFONT 系统之后，现在我已经有能力实现在开始这一研制工作时就有了的理想，即应用这些系统于《计算机程序设计艺术》中。最终，本书的全部文本都以电子文件的形式存入到我的个人计算机当中，以使它可以容易地适合于未来的印刷和显示技术。新的设置使我得以作字面上的数以千计的改进，它们是我早就想付诸实施的。

在这一新版本中，我已经字斟句酌地校阅每段文字，试图保留我原来句子的蓬勃朝气，而或许又增加上今日的深思熟虑。已经增加了几十道新习题，还有几十道老的习题也配上新的和改进了的答案。

然而，《计算机程序设计艺术》仍然在写作之中，因此本书的某些部分带有“正在施工”的图标，以表示对该部分内容尚未被更新这一事实的歉意。我的文件夹里已挤满了重要的材料，因而我打算把它们包括在最后的、光荣的第 1 卷的第 4 版中，或许在从现在开始的 15 年内吧。但我必须首先完成第 4 卷和第 5 卷，除非绝对必须，我不想把它们的出版时间拖延得更长。

准备这一新版本的大部分艰苦工作是由 Phyllis Winkler 和 Silvio Levy 完成的，他们熟练地敲入和编辑了第 2 版的文本，还有 Jeffrey Oldham，他把几乎所有原来的插图都转换成 METAPOST 的格式。我已经改正了机敏的读者们在第 2 版中发现的错误(以及某些，天哪，竟无人注意到的错误)，我已试图避免在新的内容中引进新的错误。然而，我设想，还会有某些错误存在，我要尽可能彻底地改正它们。因此我将高兴地对每一个技术、排版或历史错误的头一个发现者支付 2.56 美元。版权页上的网页包含有已向我报告的所有错误的更正[*]。

Stanford, California　　　　D.E.K

1997 年 4 月

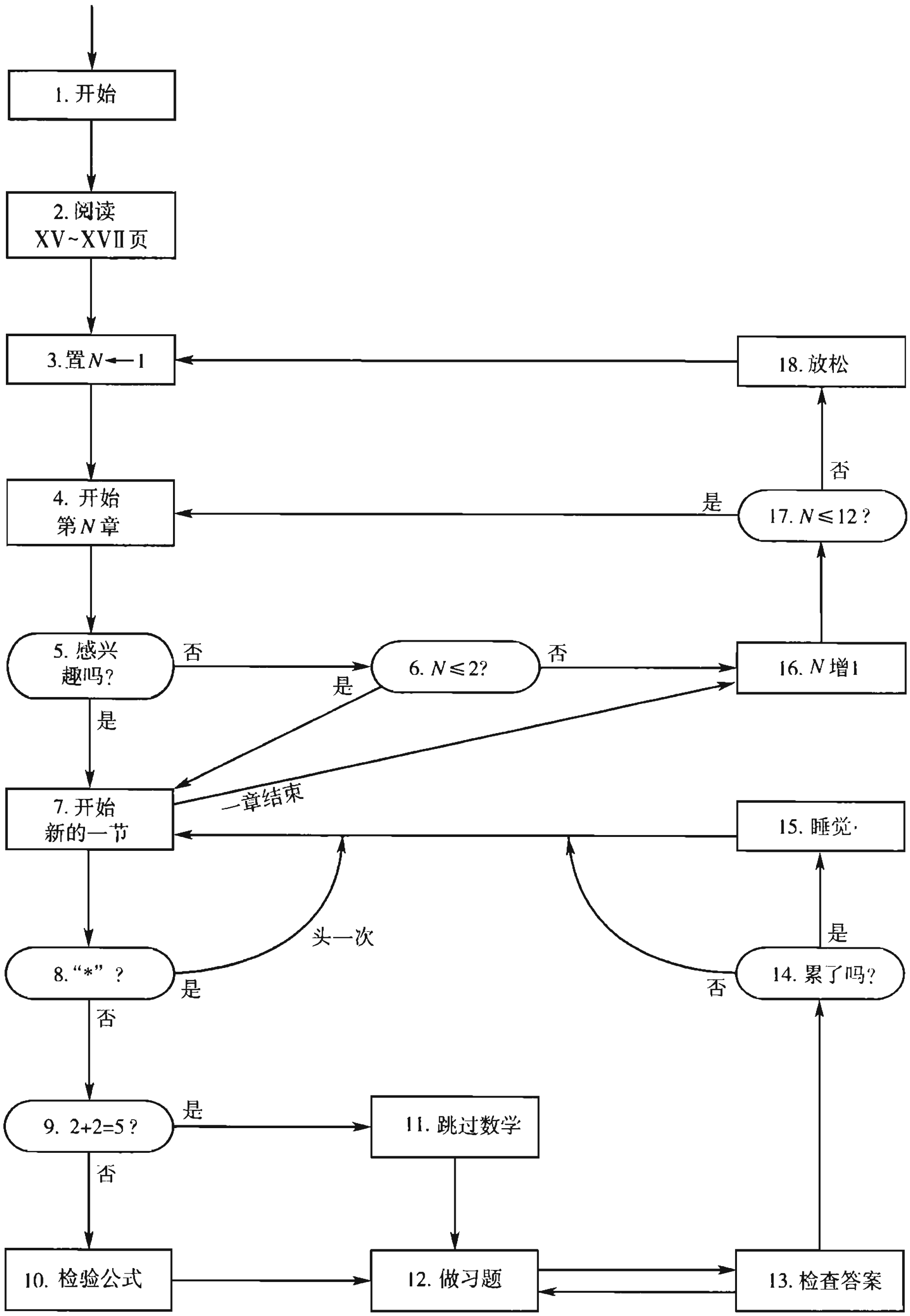
Things have changed in the past two decades.

一晃 20 年，形势变了。

—BILL GATES(1995)

---

* 中文版已按 2007 年 5 月的最新勘误表对重要疏误做了校正。网页 http://www.ndip.com.cn/computer/ 上列出了对中文版的勘误表，我将很诚挚地对发现中文版错误的读者表示感谢，但我不敢保证也支付您 2.56 美元。——本书责任编辑


1. 开始
2. 阅读
XV~XVII页
3. 置N←1
4. 开始
第N章
5. 感兴
趣吗?
否
6. N≤2?
否
16. N增1
是
是
一章结束
7. 开始
新的一节
8. "*" ?
是
头一次
否
9. 2+2=5 ?
是
11. 跳过数学
否
10. 检验公式
12. 做习题
13. 检查答案
14. 累了吗?
是
否
15. 睡觉
17. N≤12 ?
是
否
18. 放松

阅读本套书的流程图

阅读本套书的流程

1.开始,先阅读本步骤,除非你已经读过了。继续忠实地遵循这些步骤。(这一过程的一般形式及与之相应的流程图贯穿全书。)

2.阅读后文关于习题的说明。

3.置 N 等于 1。

4.开始阅读第 N 章,不要读该章开头的引语。

5.你对这章的主题感兴趣吗? 若感兴趣,转到第 7 步;若不感兴趣,转到第 6 步。

6.$N \leq 2$ 吗? 若否,转到第 16 步;若是,随便地浏览这一章。(第 1 章和第 2 章包含有重要的介绍性材料,并且也是基本的程序设计技术的综述,你应当至少浏览关于符号和关于 MIX 的部分。)

7.开始阅读本章的下一节,但若你已读完本章,就转第 16 步。

8.这一节带有“ * ”吗? 若是,则在头一次阅读时,你可略去这一节(这里包含有较为专门的课题,这些课题是有趣的,但并非是必要的);转到第 7 步。

9.你对数学有兴趣吗? 如果你对数学完全不懂,转到第 11 步;否则转到第 10 步。

10.校验这一节中所做的数学推导(并把错误之处告知作者),转到第 12 步。

11.如果这一节充满了数学计算,你最好略去这些推导不读。然而,你应当熟悉这一节的基本结果。这些基本结果通常在开始处附近叙述,或者在困难部分的末尾用*5°斜*叙述。

12.按照(你在第 2 步中读到的)关于习题的说明中给出的提示做这节中推荐的习题。

13.在你已经自认为满意地把题目做完之后,将你的答案与本书后面相应的答案进行核对(如果对于这个问题有答案的话)。也读一读你没有时间做的习题的答案。**注:**在大多数情况下,在做第 $N+1$ 题之前,先读第 N 题的答案是合理的,所以第 12 步和第 13 步通常是同时做的。

14.累了吗? 若否,返回第 7 步。

15.去睡觉,醒来后返回第 7 步。

16.N 增 1,如果 $N=3,5,7,9,11$ 或 12,开始下一卷。

17.若 N 小于或等于 12,返回第 4 步。

18.祝贺你,现在试试来说服你的朋友购买第 1 卷,并开始阅读它,而且,返回第 3 步。

Woe be to him that reads but one book.
但愿他不只读一本书而已。
—GEORGE HERBERT, *Jacula Prudentum* 144 (1640)

Le défaut unique de tous les ouvrages
c'est d'être trop longs.
它惟一的缺点就是太长了。
—VAUVENARGUES, *Réflexions* 628 (1746)

Books are a triviality. Life alone is great.
书是平凡的;只有生命才是伟大的。
—THOMAS CARLYLE, *Journal* (1839)

关于习题的说明

本套书的习题既可用于自学,也可用于课堂练习。无论是谁,如果想纯粹地通过阅读,而不将所阅读的信息应用到特定问题上,并由此牵引思考先前阅读的内容,就想学到一门学问,纵然可能,那也是很困难的。其次,对于我们自己的发现,我们总是领会得最透。因此,习题形成了这一套书的一个重要部分;我着意使这些习题含有丰富的信息,而且也尽量选择既有趣又有启发性的习题。

在许多书里,容易的习题被随机地混杂于极端困难的问题当中。有时这是不合适的,因为读者预先想要知道一道习题花多少时间——否则他们可能跳过这些习题了事。这一情况的典型例子是由 Richard Bellman 所著的*Dynamic Programming* 一书。这是一本重要的先驱性的论著,其中在某些章的末尾,在“习题和研究题”的标题下,把一组问题堆集在一起,而且一些极其平凡的问题出现在一些深入的、还未被解决的问题当中。据说,有人曾问过 Bellman 博士,怎样区分习题和研究题。他回答说:“如果你能解决它,那它就是一道习题,否则它就是一个研究题。”

但在这一类书里把研究问题和很容易的习题都包括进来确有其道理;因此,为使读者免于陷入确定哪是习题哪是研究题的困境,我给习题注明了等级分,以指出困难的程度。这些分数大体具有下列意义:

分数	解释
00	一个极其容易的问题,如果你已理解正文的内容,就可能立即做出回答。这样一道题几乎总是可以“眉头一皱”就把它做出。
10	一个简单的问题。它要求你去思考刚刚学过的内容,但绝不意味着是困难的。你应当有能力在顶多一分钟之内就把它做出。在获得解答的过程中可能要用到笔和纸。
20	一个普通的问题。它检查你对正文内容的基本理解,但你可能需要十五或二十分钟才能完整地回答它。
30	一个中等难度或中等复杂的问题。这个题目可能需要两个小时以上的工作才能令人满意地解决。或者甚至更长时间,如果电视机在开着的话。
40	确实是一个十分困难或冗长的问题。在学校里,它将适合于作为一个学期的课程设计。一个学生应当有能力在相当长的时间里来解决它,但这个解不会是平凡的。
50	就作者在编写本书时所知,这是一个还未令人满意地解决的问题,尽管已经有很多人做了尝试。如果你已经找到了这样一个问题的答案,你应当

把它写出来发表。其次,本书的作者将乐意尽快地听到关于这一解答的消息(当然,假定它是正确的)。

通过在这个"对数"尺上的内插,其它分数的意义也就清楚了。例如 17 分将表示比普通的题更为简单的一道习题。一个随后被某个读者解决了的 50 分的题在本书后来的版本中将以 45 的分数出现,而且会在互联网上的勘误表中刊出(参见版权页)。

分数除以 5 的余数表示所要求的详细工作量。因此分数是 24 的习题可能要比分数是 25 的题花费更长的时间来求解,但后者将要求更多的创造性。

作者已经认真地试图指定精确的分数,但是提出问题的人要想确切地知道,所提问题对求解的另一个人难度如何,肯定是困难的;而且每个人都会有较其他人更为适应的问题类型。只能希望这些分数较好地反映了习题的困难程度,希望读者把它们当做一般的导引,而不应作为绝对的指标。

本书是为具有不同程度的数学训练和素养的读者写的;因此有些习题是特意为有更多数学基础的读者提供的。如果一道题的出题动机或涉及的数学概念超越了主要兴趣仅仅是算法编程本身的读者应掌握的程度,则在分数前边加上 *M*。如果一道习题的答案必须涉及在本书中未予提供的微积分或其它高等数学知识,则对这道题标以"*HM*"的字母。"*HM*"标记并不必定意味着困难。

某些习题的前边标有三角符"▶",这说明是特别有启发性的问题,因而特别予以推荐。当然,并不期望读者/学生来求解所有的习题,所以标出了那些看起来最有价值的部分(这并不意味着贬低其它习题!)。每个读者至少应该尝试去解分数是 10 或者更低的所有问题;三角符可以帮助指出应该对具有较高分数的哪些问题予以优先考虑。

在答案部分中已经列出大多数习题的解答,请明智地使用它们。在你已真正地做出努力亲自解决问题之前,不要去翻答案,除非你确实没有时间来做这一特定的习题。在获得了你自己的解答或者对该题做了郑重的尝试之后,你可能会感到答案是有启发和有帮助的。给出的解答通常十分简短,作者认为你已经认真地通过自己的方法做过求解尝试,因而略去了其细节。有时解答所提供的信息比所要求的为少,但通常都是更多的。十分可能,你有比这里给的更好的答案,或者在答案中你发现一个错误。在这样的情况下,作者将乐意知道详细的情况。在本书以后的版本中将在适当地方刊登出这些改进了的答案以及提供这些解的人的姓名。

当做一道习题时,一般地你可以使用前边习题的答案,除非明确地禁止这样做。在对习题打分时,已经考虑到这一点了,因此很可能第 $n+1$ 题的分数比第 n 题还要低,尽管它把第 n 题的结果作为一个特殊情况包括进来。

代码	含义		
		00	立即可解的
		10	简单的(一分钟)
▶	特别推荐的	*20*	一般水平(一刻钟)
M	面向数学的	*30*	难度适中
HM	要求"高等数学"的	*40*	学期设计
		50	研究题

习 题

▶ 1.[*00*] 分数“*M20*”意味着什么?

2.[*10*] 一本教科书中的习题对读者来说有什么价值?

3.[*14*] 证明 $13^3 = 2197$ 并推广你的答案。[这是一类令人讨厌的问题,作者已试图避免它。]

4.[*HM45*] 证明当 n 是整数且 $n > 2$ 时,方程 $x^n + y^n = z^n$ 无正整数解 x, y, z。

We can face our problem.
We can arrange such facts as we have
with order and method.
我们能够面对我们的问题。
我们可以搞定它们,
因为我们有规则和方法。

—HERCULE POIROT、*Murder on the Orient Express* (1934)

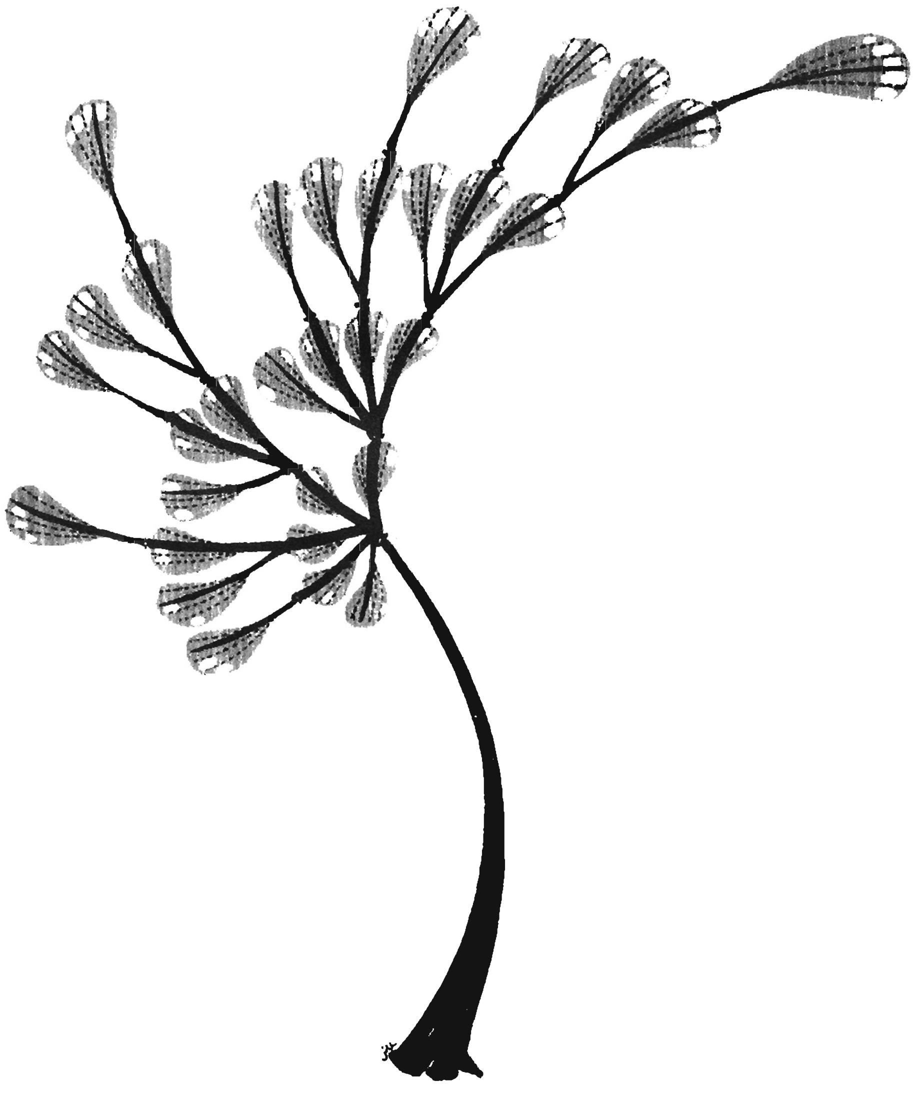
目　　录

第1章　基本概念

1.1　算法 …… 1
1.2　数学准备 …… 9
　1.2.1　数学归纳法 …… 9
　1.2.2　数,幂和对数 …… 19
　1.2.3　和与积 …… 24
　1.2.4　整数函数和初等数论 …… 35
　1.2.5　排列和阶乘 …… 41
　1.2.6　二项式系数 …… 48
　1.2.7　调和数 …… 72
　1.2.8　斐波那契数 …… 76
　1.2.9　生成函数 …… 83
　1.2.10　一个算法的分析 …… 93
　*1.2.11　渐近表示 …… 102
　　*1.2.11.1　O 符号 …… 103
　　*1.2.11.2　欧拉求和公式 …… 107
　　*1.2.11.3　一些近似计算 …… 112
1.3　MIX …… 120
　1.3.1　MIX的描述 …… 120
　1.3.2　MIX汇编语言 …… 140
　1.3.3　对排列的应用 …… 157
1.4　某些基本程序设计技术 …… 178
　1.4.1　子程序 …… 178
　1.4.2　共行程序 …… 185
　1.4.3　解释性程序 …… 191
　　1.4.3.1　一个MIX模拟程序 …… 193
　　*1.4.3.2　跟踪程序 …… 203
　1.4.4　输入和输出 …… 206
　1.4.5　历史和文献 …… 219

第2章 信息结构

2.1 引论 …… 222
2.2 线性表 …… 227
  2.2.1 栈,队列和双端队列 …… 227
  2.2.2 顺序分配 …… 232
  2.2.3 链接分配 …… 241
  2.2.4 循环表 …… 258
  2.2.5 双重链接表 …… 265
  2.2.6 数组和正交表 …… 284
2.3 树 …… 293
  2.3.1 遍历二叉树 …… 302
  2.3.2 树的二叉树表示 …… 316
  2.3.3 树的其它表示 …… 331
  2.3.4 树的基本数学性质 …… 343
    2.3.4.1 自由树 …… 343
    2.3.4.2 有向树 …… 351
    *2.3.4.3 “无穷性引理” …… 360
    *2.3.4.4 树的枚举 …… 364
    2.3.4.5 通路长度 …… 376
    *2.3.4.6 历史和文献 …… 382
  2.3.5 列表和废料收集 …… 384
2.4 多重链接结构 …… 399
2.5 动态存储分配 …… 409
2.6 历史和文献 …… 429
习题答案 …… 436
附录 A 数值数量表 …… 588
附录 B 记号索引 …… 592
索引与词汇表 …… 596



第 1 章 基本概念

Many persons who are not conversant with mathematical studies imagine that because the business of [Babbage's Analytical Engine] is to give its results in numerical notation, the nature of its processes must consequently be arithmetical and numerical, rather than algebraical and analytical. This is an error. The engine can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols; and in fact it might bring out its results in algebraical notation, were provisions made accordingly.

许多对于数学研究不熟悉的人想像，
由于[巴贝奇分析机的]工作是以数值符号给出它的结果，
因此它的过程的本质也必然是算术的和数值的，而不是代数和分析的。
这是错误的。分析机能够把它的数值量安排和组合成
就如同它们是字母或任何其它一般符号那样，
而且事实上它可以以代数符号来给出它的结果，假如相应地采取措施的话。

—AUGUSTA ADA, Countess of Lovelace (1843)

Practice yourself, for heaven's sake, in little things;
and thence proceed to greater.
看在上帝的份上，先从小事做起吧；
而后才能做更大的事。

—EPICTETUS (*Discourses* IV.i)

1.1 算 法

算法是贯穿在所有计算机程序设计中的一个基本概念，所以我们应当首先细心地分析这一概念。

"算法"(Algorithm)一词本身就是十分有趣的。初看起来，好像是某人打算要写"对数"(Logarithm)一词，但却把头四个字母顺序写乱了。这个词直到 1957 年在《韦伯斯特新世界词典》(*Webster's New World Dictionary*)中还未出现；我们只能找到带有它的古代含义的较老形式的"算术"(algorism)，指的是用阿拉伯数字进行算术运算的过程。在中世纪时，珠算家用算盘进行计算，而算术家用算术进行计算。文艺复兴时期，对这个词

的来源已经拿不准了。早期的语言学家试图猜测说，它是 *algiros*［费力的］+ *arithmos*［数字］组合起来而派生出来的；但是另外一些人则不以为然，认为这个词是从“King Algor of Castile”派生而来的。最后，数学史家发现了“algorism”一词的真实来源：它来自于一部著名的波斯课本的作者的名字 Abū ‘Abd Allāh Muhammad ibn Mūsā al－Khwārizmī（约公元 825 年）——从字面上看，是“Abdullah，Mohammad 的父亲，Moses 的儿子，Khwārizm 地方的人”。中亚的咸海（Aral Sea）曾经叫做科瓦茨米湖（Lake Khwārizm），而 Khwārizm 区域位于该海南部的阿姆河（Amu River）流域。Al-Khwārizmī 写了名著 *Kitāb al-jabr Wa'l-muqābala*（《复原和化简的规则》）；另一个词“代数”（algebra），是从他的书的标题引出来的，该书是对线性和二次方程的解的系统研究。［关于 al-Khwārizmī 的生活和著述的评述，见 H. Zemanek, *Lecture Notes in Computer Science* **122**（1981），1～81。］

逐渐地，algorism 的形式和意义就变得面目全非了；如《牛津英语辞典》（*Oxford English Dictionary*）所说明的，这个词“经历了许多伪词源学的滥用，包括最新的 algorithm。在这些词源学中它被学术性地同具有希腊根源的词 arithmetic（算术）混淆”。考虑到人们已经把这个词原始的派生过程忘记了这一事实，从“algorism”变成为“algorithm”就不难理解了。一本早期的德国数学词典 *Vollständiges mathematisches Lexicon*（Leipzig：1947）给出了 Algorithmus（算法）一词的如下定义：“在这个名称之下，组合了四种类型的算术计算的概念，即加法、乘法、减法和除法。”拉丁短语 algorithmus infinitesimalis（无限小算法）在当时就用来表示“莱布尼茨（Leibniz）所发明的以无限小量进行计算的方法”。

到 1950 年，algorithm 一词经常同欧几里得算法联系在一起。这个算法就是在欧几里得的 *Elements*（第 7 卷，命题 1 和 2）中所阐述的求两个数的最大公因子的过程。在这里给出欧几里得算法将是有益的。

算法 E（欧几里得算法）　给定两个正整数 m 和 n，求它们的最大公因子，即能够同时整除 m 和 n 的最大正整数。

E1.［求余数］以 n 除 m 并令 r 为所得余数。（我们将有 $0 \leq r < n$。）

E2.［余数为零？］若 $r = 0$，算法结束，n 即为答案。

E3.［减少］置 $m \leftarrow n$，$n \leftarrow r$，并返回步骤 E1。▌

当然，欧几里得并非就是以这样的方式给出他的算法的，上面的格式充分显示了在给出本书中所有算法时贯彻始终的风格。

我们对于所讨论的每一个算法。都给出了一个标记字母（例如在上边的例子中的 E），并用在这个字母后面接上一个数字（E1，E2，E3）来标记算法的步骤。书中各章分为编了号的节；在一节之内的算法仅用字母来标记；但当在其它节中引用这些算法时，则附以相应的节号。例如，我们现在是在第 1.1 节；在这节内，欧几里得算法就叫做算法 E，而在后边的节引用它时，它就叫做算法 1.1E。

一个算法的每一步，例如上边的步骤 E1，以一个方括号中的一个短语开始，它尽可能简短地概括这一步的主要内容。这个短语通常也出现在一个相应的流程图中，例如图 1，使得读者能更容易地想像这个算法。

在概括性的短语之后是有待执行的某个动作或有待做出的某个判断的文字和符号

的叙述。也可能出现带有括号的注释，如同在步骤 E1 中的第二句。注释是作为该步的说明性信息而被包括进来的，通常指出在该步时变量的不变的特征或当前的目标，它们并不描述属于此算法的动作，而仅用于帮助读者理解。

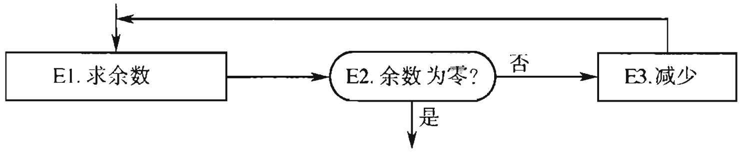


图 1 算法 E 的流程图

步骤 E3 中的箭头"←"是最重要的替代运算符，有时叫做赋值或代换。"$m \leftarrow n$"意思是变量 m 的值要被变量 n 的当前值所代替。当算法 E 开始时，m 和 n 的值是原来给定的数；但当它结束时，一般说来，这些变量将有不同的值。箭头用来区分替代运算和相等关系，我们将不说"置 $m = n$"，但或许我们将问"$m = n$ 吗?"，"="符号表示可被测试的一个条件，而"←"符号表示可被实施的一个动作。n 增加 1 这一运算由"$n \leftarrow n + 1$"（读作"n 为 $n + 1$ 所代替"或者"n 变成 $n + 1$"）表示。一般地说，"变量←公式"指的是该公式要用出现于其中的任何变量当前的值加以计算；而后计算结果应替换出现于箭头左边的变量以前的值。未曾受过计算机工作训练的人们有时倾向于说"n 变成 $n + 1$"，并且写"$n \rightarrow n + 1$"来表示 n 增加 1 的运算。这种符号只会导致混乱，因为它同标准的约定相冲突，因而应予避免。

注意在步骤 E3 中的动作顺序是重要的："置 $m \leftarrow n$，$n \leftarrow r$"十分不同于"置 $n \leftarrow r$，$m \leftarrow n$"，因为后者将意味着在 n 以前的值可被用于置给 m 之前，它就已失去了。因此后一序列等价于"置 $n \leftarrow r$，$m \leftarrow r$"。当要把若干个变量都置成等于同一个数量时，我们可以使用多个箭头：例如："$n \leftarrow r$，$m \leftarrow r$"可以写成为"$n \leftarrow m \leftarrow r$"。为了交换两个变量的值，我们可以写"交换 $m \leftrightarrow n$"；这一动作也可通过使用一个新变量 t 并写"置 $t \leftarrow m$，$m \leftarrow n$，$n \leftarrow t$"来确定。

一个算法从编号最低的步骤，通常就是步骤 1 开始，它顺序地执行随后的步骤，除非另有说明。在步骤 E3 中，命令式的"返回步骤 E1"以明确的方式说明计算的顺序。在步骤 E2 中，动作前边有一个条件"若 $r = 0$"；因此，如果 $r \neq 0$，这个句子剩余部分不适用，因而就无动作发生。我们甚至可以加上多余的句子，"如果 $r \neq 0$，转到步骤 E3"。

出现在步骤 E3 末尾的粗的垂直黑线"▌"用来表示一个算法的结束和正文的继续。

我们实际上已经讨论了本书的算法使用的所有符号的约定，除了用于表示"带下标"(subscripted)或"带中标"(indexed)* 的项之外，它们是一个有序数组的元素。假设我们有 n 个量，$v_1, v_2, \cdots, v_n$；通常我们用 $v[j]$ 而不是 v_j 表示其中第 j 个元素。类似地，相对于像 a_{ij} 这样的双重下标的符号，我们有时更偏爱使用 $a[i,j]$。有时使用多字母名称

* 后文仍用"下标"作为 index 的译名。——本书责任编辑

表示变量，通常是以大写字母表示。因此 TEMP 可能是用于临时保持一个计算值的变量名，PRIME[K]可能表示第 K 个素数，等等。

关于算法的形式就讲这些；现在让我们执行一个算法。应当立即提到，读者不应期望像读小说一样来读算法，如果抱有这样的想法，就很难理解算法的进程。算法应是可信的，而学习算法的最好方法是尝试它。读者应当笔纸不离手，并在课文中遇到算法时，立即运行算法的一个例子。通常我们都给出一个有效实例的要点，要不读者也能很容易地想出一个例子。这是理解一个给定算法的简单而轻松的方法，所有其它方法一般说来都是不成功的。

因此让我们来考察算法 E 的例子。假设给定 $m = 119$ 和 $n = 544$；我们现在已准备好从步骤 E1 开始(读者应当跟着我们的“现场评述”循着算法进行)。在此情况下，m 除以 n 十分简单，简直太简单了。因为商为 0 而余数是 119。因此，$r \leftarrow 119$。我们进行到步骤 E2。而由于 $r \neq 0$，因此无动作出现。在步骤 E3 中，我们置 $m \leftarrow 544$，$n \leftarrow 119$。显然，如果原来 $m < n$，步骤 E1 中的商将总为 0，因此算法在这种相当讨厌的方式下总是进行 m 和 n 的交换。我们可以增加一个新的步骤：

E0. [确保 $m \geqslant n$] 如果 $m < n$，交换 $m \leftrightarrow n$。

这在算法中不产生任何实质性的改变，只是稍稍增加了算法的长度，却在大约一半情况下减少了其运行时间。

回到步骤 E1，我们发现 $544/119 = 4 + 68/119$，因此 $r \leftarrow 68$。E2 再一次不起作用。因而在步骤 E3，我们置 $m \leftarrow 119$，$n \leftarrow 68$。下一轮置 $r \leftarrow 51$，最终 $m \leftarrow 68$，$n \leftarrow 51$。其次 $r \leftarrow 17$ 且 $m \leftarrow 51$，$n \leftarrow 17$。最后，当 51 为 17 整除时，置 $r \leftarrow 0$，所以算法在步骤 E2 结束。119 和 544 的最大公因子是 17。

这就是一个算法。算法的现代意义十分类似于菜谱(recipe)，进程(process)，方法(method)，技术(technique)，过程(procedure)，例程(routine)，仪式程序(rigmarole)，但“算法”一词包含有那么一点不同的东西。一个算法只不过是一组有穷的规则，这些规则给出求解特定类型问题的运算序列；但除此之外，一个算法还有五个重要特征：

1)有限性。一个算法在有限步骤之后必然要终止。算法 E 满足这一条件，因为在步骤 E1 之后，r 的值小于 n；所以如果 $r \neq 0$，在下次遇到步骤 E1 时，n 的值减少。一个正整数的递减序列必然最终会终止，因此对于任何给定的 n 的原始值，步骤 E1 仅被执行有限次。然而请注意，步骤数可以变成任意地大；选择某些巨大的 m 和 n 将引起步骤 E1 被执行超过百万次。

(一个具有算法的所有其它特征，只是可能缺少有限性的过程，可以叫做一个计算方法(computational method)。欧几里得原来给出的不仅仅是求数的最大公因子的算法，而且也是求两个线段长度的“最大公共量度”的非常类似的几何构造。假如给定的长度是不可约数的话，这是一个不终止的计算方法。不终止的计算方法的另一个例子是往复过程(reactive process)，它总是不断地与它的环境互相作用。)

2)确定性。一个算法的每个步骤都必须精确地定义；要执行的动作每一步都必须严格地和无歧义地描述清楚。本书的算法有希望满足这一准则，但它们是以英语描述

的,因而也有可能读者未能准确地理解作者的意图。为了克服这个困难,我们设计了用于描述算法的形式地定义的程序设计语言或计算机语言,其中每一个语句都有非常确定的意义。本书的许多算法都将以英语和以一种计算机语言给出,在一种计算机语言下一个计算方法的表达就叫做一个程序(program)。

在算法 E 中,应用于步骤 E1 的确定性准则意味着读者应当确切地理解 m 除以 n 指的是什么以及余数是什么。事实上,如果 m 和 n 不是正整数,那么关于它意味着什么是没有普遍接受的认识的; -8 除以 $-\pi$ 的余数是什么? 59/13 除以 0 的余数是什么? 因此,确定性的准则意味着我们必须确保每当步骤 E1 被执行时, m 和 n 的值总是正整数。由假设,这在开始时为真;而在步骤 E1 之后,如果我们到达步骤 E3, r 必定是非零的非负整数。因此 m 和 n 确实如同所要求那样是正整数。

3)输入。一个算法有零个或多个输入,此即在算法开始之前最初赋给它的量,或者当算法运行时动态地赋给它的量。这些输入是从特定的对象集合取出的。例如,在算法 E 中,有两个输入,即 m 和 n,两者都是从正整数集合取出来的。

4)输出。一个算法有一个或多个输出:和输入有特定关系的量。算法 E 有一个输出,即步骤 E2 中的 n,即两个输入的最大公因子。

(这个数确实是最大公因子.我们可容易地证明如下:在步骤 E1 后,对于某个整数 q,我们有

$$m = qn + r$$

如果 $r=0$,则 m 是 n 的倍数,显然在这样一种情况下, n 是 m 和 n 的最大公因子。如果 $r\neq 0$,注意整除 m 和 n 的任何数必定也整除 $m - qn = r$,因此整除 n 和 r 的任何数必定整除 $qn + r = m$;所以 $\{m, n\}$ 的公因子集合和 $\{n, r\}$ 的公因子集合是一样的。特别地, $\{m, n\}$ 的最大公因子和 $\{n, r\}$ 的最大公因子是一样的。因此步骤 E3 不改变原来问题的答案。)

5)能行性。一个算法一般可以认为是能行的(或称有效的),其含义是指它的所有运算必须是充分基本的,因而原则上人们用笔和纸都可在有限时间内精确地完成它们。算法 E 仅仅使用以一个正整数来除另一个正整数,检查一个整数是否为零,以及置一个变量的值为另一个变量的值等运算。这些运算是能行的:因为整数在纸上可以以有限的方式表示,而且至少有一种方法("除法算法")来进行一个整数除另一个整数的运算。但是,如果所涉及的值是由无穷小数展开确定的任意实数,同样的运算就不是能行的。如果这些值是物理线段的长度(它们不能精确地确定),那么运算也不是能行的。不是能行的步骤的另一个例子是,"如果 4 是使得方程 $w^n + x^n + y^n = z^n$ 在正整数的 w, x, y 和 z 中有解的最大整数 n,则转到步骤 E4"。这样一个语句在某人成功地构造出一个算法确定 4 是否具有所描述性质的最大整数之前,将不是一个能行的运算。

让我们试图来对算法的概念同菜谱的概念做一下比较。一个菜谱大抵具有有限性(尽管有人说,心急水不沸),输入(鸡蛋、面粉等)以及输出(速食便餐等)等性质,但是众所周知它缺少确定性。经常有这样的情况,菜谱的指导是不确定的:"加少许盐。""少许"被定义为"少于 1/8 茶匙",或许盐是明确地定义的;然而,把盐加在哪儿呢? 在顶上

还是在边上？像“轻微地搅拌直到混合物变碎为止”或者“在小深平底锅中把法国白兰地酒加热”这样一些指导对于训练有素的厨师可能是十分适当的说明，但是一个算法必须被描述到这样一种程度，就是即使一部计算机也能遵循它。尽管如此，一个计算机程序员通过学习一本好的菜谱书也可以学到很多东西。(本书作者事实上几乎很难拒绝把本卷书取名为“程序员的菜谱”这样一个具有诱惑力的名字。也许某一天他将试图写一本叫做“厨房的算法”的书。)

我们应当说明，就实用而言，有限性的限制实际上是不够强的。一个有用的算法不只要求步骤有限，而且要求非常有限的、合理的步骤数。例如，有一个算法要确定，如果不出错，执白棋者是否总能下赢一盘棋(见习题 2.2.3-28)。这个算法能解决成千上万人强烈地感兴趣的一个问题；但是可以稳赢地打赌，在我们的一生中，我们都不会知道这个问题的答案。这个算法要求大得难以置信的执行时间，即使它是有限的。有关某些有限数的讨论，请参看第 8 章。这些数是如此之大，实际上已经超出了我们的理解能力。

在实践中，我们不仅要算法，而且还要在某种不明确定义的意义下的*好*算法。好的一个准则是用于执行算法的时间长度，这可以借助于执行每个步骤的次数来表示。其它准则有算法对于不同类型的计算机的适应性、简单性以及典雅性等等。

我们经常面对着同一个问题的若干个算法，因而我们必须判断哪一个最好。这就导致了*算法分析*这一极为有趣和十分重要的领域：给定一个算法，我们要确定它的性能特征。

例如，让我们从这一观点出发来考虑欧几里得算法。假设我们提问，“假定 n 的值已知，而允许 m 取遍所有的正整数，算法 E 的步骤 E1 将被执行的平均次数 T_n 是多少?”首先，我们需要检查这个问题是否确有一个有意义的答案，因为我们是在试图对于 m 的无穷多个选择取平均值。但是显然，在头一次执行步骤 E1 之后，有关的只不过是 m 除以 n 之后的余数。因此，为了找出 T_n 我们所要做的一切就是对于 $m=1$，$m=2,\cdots,m=n$ 来尝试算法，计算步骤 E1 被执行的总次数，并除以 n。

现在重要的问题是确定 T_n 的*性质*。比如说，它近似地等于 $(1/3)n$？还是等于 $\sqrt{n}$ 呢？事实上，这一问题的解答是一个极其困难而又令人着迷的数学问题，而且还未被彻底解决，我们将在 4.5.3 小节中对它作更详细的考察。对于很大的 n 值，有可能证明，T_n 近似于 $(12(\ln 2)/\pi^2)\ln n$，即它与 n 的自然对数成正比，同时还带有未曾即时猜出的比例常数！有关欧几里得算法以及计算最大公因子的其它方法的进一步的细节，请参见 4.5.2 小节。

*算法分析*是作者希望用于描述这样一类研究的名称。它的一般思想是选取一个特定的算法，然后来确定它的定量的特性；有时我们还要研究一个算法在某种意义下是否最优。至于*算法理论*则完全是另一个学科，它主要讨论计算特定量的有效算法的存在或不存在。

迄今为止，我们有关算法的讨论是相当不精确的。因而，面向数学的读者理所当然会认为，前边的评述是关于算法的任何理论的非常不坚实的基础。因此，我们以一个方法的简要表达来结束这一节，通过这一方法，借助于数学的集合论，可以为算法的概念

打下坚实的基础。我们形式地定义一个计算方法为一个四元组(Q,I,Ω,f),其中 Q 是包含子集 I 和 Ω 的一个集合,而 f 是从 Q 到其自身的一个函数。其次,f 应保持 Ω 逐点不动;即,对于 Ω 中的所有元素 q,$f(q)$应等于 q。四个量 Q,I,Ω,f 分别用于表示计算的状态,输入,输出以及计算规则。集合 I 中的每一个输入 x 定义一个计算序列 $x_0,x_1,x_2,\cdots$如下:

$$x_0=x \quad 及 \quad 对于\ k\geqslant 0, \quad x_{k+1}=f(x_k) \tag{1}$$

如果 k 是使 x_k 在 Ω 中为最小的整数,就说计算序列在 k 步中终止,而且在此种情况下,说从 x 产生出输出 x_k。(注意,如果 x_k 在 Ω 中,则 x_{k+1}也是。因为在这样的情况下,$x_{k+1}=x_k$。)某些计算序列可能永不终止;一个*算法*是对于 I 中所有的 x 在有限多步中终止的计算方法。

例如,算法 E 可以用这些术语表述如下:令 Q 为所有单点(n),所有有序偶(m,n)以及所有有序四元组$(m,n,r,1)$,$(m,n,r,2)$以及$(m,n,p,3)$的集合,其中 m,n,p 是正整数,r 是一个非负整数。令 I 是所有数偶(m,n)的子集,并令 Ω 是所有单点(n)的子集。定义 f 如下:

$$\begin{aligned}
&f((m,n))=(m,n,0,1);\quad f((n))=(n);\\
&f((m,n,r,1))=(m,n,m\ 除以\ n\ 的余数,2);\\
&f((m,n,r,2))=(n),如果\ r=0,\quad 否则(m,n,r,3);\\
&f((m,n,p,3))=(n,p,p,1)
\end{aligned} \tag{2}$$

这一记法同算法 E 之间的对应是明显的。)

这个算法概念的表述并不包括前边提到的能行性的限制。例如,Q 可以表示用笔和纸的方法不能计算的无穷序列,或者 f 可能涉及仅靠大脑并非总能实现的运算。如果我们希望对算法的概念加上限制,使得它仅涉及初等的运算,我们就可以对 Q,I,Ω 和 f 加上一些限制。比如说,令 A 是字母的有限集合,并令 A^* 是 A 上的所有串的集合(即所有有序序列 $x_1x_2\cdots x_n$ 的集合,其中 $n\geqslant 0$ 且对于 $1\leqslant j\leqslant n$,x_j 在 A 中)。想法是对计算的状态进行编码使得它们可以由 A^* 的串来表示。现在设 N 是非负整数,而 Q 是所有(σ,j)的集合,其中 σ 在 A^* 中,而 j 是一个整数,$0\leqslant j\leqslant N$;设 I 是 $j=0$ 的 Q 的子集,设 Ω 是 $j=N$ 的 Q 的子集。如果 θ 和 σ 都是 A^* 中的串,我们说如果对于串 α 和 ω,σ 有 $\alpha\theta\omega$ 的形式,则 θ 在 σ 中出现。为了完成我们的定义,令 f 是对于 $0\leqslant j<N$,由串 θ_j,ϕ_j 和整数 a_j,b_j所定义的下列类型的一个函数:

$$\begin{aligned}
&f((\sigma,j))=(\sigma,a_j),\ 如果\ \theta_j\ 不在\ \sigma\ 中出现;\\
&f((\sigma,j))=(\alpha\phi_j\omega,b_j),\ 如果\ \alpha\ 是使\ \sigma=\alpha\,\theta_j\,\omega\ 最短的串;\\
&f((\sigma,N))=(\sigma,N)
\end{aligned} \tag{3}$$

这样一个计算方法显然是能行的,而且经验表明,它也足够强有力来做我们手工能做的任何事情。还有许多实际上等价的其它方法表述一个能行的计算方法的概念(例如,使用图灵机)。上边的表述实质上和 A. A. Markov 在他的书 *The Theory of Algorithms* [*Trudy Mat. Inst. Akad. Nauk* **42** (1954),1~376]中给出的阐述是一样的。该书后来由 N.M.Nagorny 修改和扩充(Moscow:Nauka,1984;英文版,Dordrecht:Kluwer,1988)。

习　题

1.[*10*] 正文中说明了如何使用替代符号，通过设置 $t \leftarrow m$，$m \leftarrow n$，$n \leftarrow t$ 来交换变量 m 和 n 的值。说明如何通过一系列的替代，把四个变量(a,b,c,d)的值重新安排成(b,c,d,a)。换言之，a 的新值是 b 原来的值，等等。试使用最少的替代次数。

2.[*15*] 证明，在步骤 E1 开始时，m 总是大于 n 的，除了这一步头一次出现时可能出现相反情况外。

3.[*20*] (为了提高效率)改变算法 E 使得像“$m \leftarrow n$”这样平凡的替代运算都加以避免。以算法 E 的风格来写出这个新算法，并称之为算法 F。

4.[*16*] 2166 和 6099 的最大公因子是多少?

▶ 5.[*12*] 证明出现于前言后的“阅读本套书的流程”，按我们给出的五条准则的三条，实际上不是一个真正的算法！并指出它和算法 E 之间格式上的差别。

6.[*20*] 当 $n=5$ 时，执行步骤 E1 的平均次数 T_5 是多少?

▶ 7.[*M21*] 假设 m 已知而允许 n 取遍所有的正整数；令 U_m 是在算法 E 中执行步骤 E1 的平均次数。证明 U_m 是明确定义的。U_m 同 T_m 有什么关系吗?

8.[*M25*] 通过描述等式(3)中的 θ_j,ϕ_j,a_j,b_j，给出计算正整数 m 和 n 的最大公因子的一个“能行的”形式算法。令输入由串 $a^m b^n$ 表示，即 m 个 a 后边跟着 n 个 b。试尽可能简单地给出你的解。[提示：使用算法 E，但代替步骤 E1 中的除法，置 $r \leftarrow |m-n|$，$n \leftarrow \min(m,n)$。]

▶ 9.[*M30*] 假设 $C_1=(Q_1,I_1,\Omega_1,f_1)$和 $C_2=(Q_2,I_2,\Omega_2,f_2)$是计算方法。例如，$C_1$ 可代表等式(2)中的算法 E，但 m 和 n 在数量上要加限制，而 C_2 可代表实现算法 E 的计算机程序。(因此 Q_2 可以是机器的所有状态的集合，即它的内存和寄存器所有可能的配置；f_2 可以是单个机器动作的定义；I_2 可以是初始状态，包括确定最大公因子的程序以及 m 和 n 的值。)

试表述“C_2 是 C_1 的表示”或“C_2 模拟 C_1”的概念的集合论定义。这在直观上意味着 C_1 的任何计算序列都由 C_2 加以模拟，除 C_2 在计算中采用更多的步骤以及在它的状态中它可能保留更多的信息外。(我们因此得到对于“程序 X 是算法 Y 的一个实现”这一语句的一个严格的解释。)

1.2 数学准备

在这一节中,我们将研究在整个《计算机程序设计艺术》中出现的数学记号或符号,而且我们将推导要被反复地使用的若干基本公式。即便读者对于更复杂的数学推导不感兴趣,他至少也应该熟悉各种公式的含义,以便能够使用这些推导的结果。

在本书中使用数学符号的目的有二,一是描述一个算法的组成部分,一是分析一个算法的性能特征。如同在上一节说明的那样,用于描述算法的记号是十分简单的,但当分析算法的性能时,我们需要使用其它更专门的记号。

我们将讨论的大多数算法都伴随有数学计算,这些计算确定算法预期的运行速度。这些计算几乎取自于每一个数学分支,因而为展开本书不同地方引用的所有数学概念,将需要单独一本书。然而,大多数的计算都可以通过大学代数的知识来完成,而具有初等微积分知识的读者将能够理解出现的几乎所有数学。有时我们将需要使用复变函数论、群论、数论、概率论等等更深入的结果;在这种情况下,我们将以初等的方式来对课题进行说明,如果可能的话,或者将给出其它信息来源的参考。

在算法分析中涉及的数学技术通常有着独特的风味。例如,我们将经常对有理数进行有限求和,或者求解递推关系。传统上这样的课题在数学课程中仅作肤浅的讨论,因此在随后的几个小节里我们不仅对于要加以定义的符号进行深入的讨论,而且还将深入地说明对我们来说最为有用的计算类型和技术。

重要说明:尽管下边的几个小节提供了同计算机算法相联系所需要的数学技巧方面相当广泛的训练,但大多数读者开初将看不见这些材料同计算机程序设计之间很强的联系(1.2.1 小节除外)。读者可以仔细地有选择地阅读以下各小节,同时请相信作者的断言,即我们这里讨论的这些课题确实是非常有关系的;但是作为促动因素,开始*时稍微浏览这一节,而后*(在见到未来各章中许多技术的应用之后)*再对它们作更深入的研究*大概是可取的。如果在开始阅读本书时就花费太多时间在学习这些内容上,那一个人就很可能老也开始不了计算机程序设计的课题!然而,每个读者至少应该熟悉这些小节的一般内容,并且应当尝试来解少量习题,即使在刚开始阅读时。1.2.10 小节应受到特别注意,因为它是后边建立的大多数理论内容的出发点。1.2 节之后的 1.3 节,突然离开"纯粹数学"的王国而进入"纯粹的计算机程序设计"。

有关后边许多内容的展开和更为从容的介绍,可在 Graham, Knuth 和 Patashnik 所写的书*Concrete Mathematics* 第 2 版(Reading, Mass.: Addison-Wesley, 1994)中找到,以后当我们需要引用该书时,就简单地叫它做*CMath*。

1.2.1 数学归纳法

令 $P(n)$为关于整数 n 的某个命题;例如,$P(n)$可以是"n 乘($n+3$)是一个偶数"或者"如果 $n\geq 10$,则 $2^n > n^3$"。假设我们要 *证明,对于所有正整数 n, $P(n)$为真*。实现这一点的一个重要方法是:

a) 给出 $P(1)$为真的证明。

b) 给出“如果所有 $P(1),P(2),\cdots,P(n)$都为真,则 $P(n+1)$ 也为真”的证明;这个证明应对任何正整数 n 都为真。

作为一个例子,考虑自古代就已经有许多人独立地发现的下列等式序列:

$$1 = 1^2$$
$$1+3 = 2^2$$
$$1+3+5 = 3^2$$
$$1+3+5+7 = 4^2$$
$$1+3+5+7+9 = 5^2 \tag{1}$$

我们可以把一般的性质表述如下:

$$1+3+\cdots+(2n-1) = n^2 \tag{2}$$

暂时,就让我们称之为 $P(n)$;我们希望证明,对于所有正的 n,$P(n)$为真。遵循上边概述的步骤,我们有:

a)“$P(1)$为真,因为 $1=1^2$。”

b)“如果 $P(1),\cdots,P(n)$都为真,特别是 $P(n)$为真,则等式(2)成立;两边都加上 $2n+1$ 我们得到 $1+3+\cdots+(2n-1)+(2n+1)=n^2+2n+1=(n+1)^2$,这就证明 $P(n+1)$也为真。”

我们可以把这个方法看做一个*算法的证明步骤*。事实上下列算法产生对任意正整数 n,$P(n)$的一个证明,假定上边的步骤 a)和 b)已经证实的话。

算法 I(*构造一个证明*)　给定一个正整数 n,这个算法将输出 $P(n)$为真的一个证明。

I1. [证明 $P(1)$] 置 $k\leftarrow 1$,而且按照 a),输出 $P(1)$的一个证明。

I2. [$k=n$?] 如果 $k=n$,算法终止;所要求的证明已被输出。

I3. [证明 $P(k+1)$] 按照 b),输出“如果所有的 $P(1),\cdots,P(k)$都为真,则 $P(k+1)$为真”的证明,也输出“我们已经证明 $P(1),\cdots,P(k)$;因此 $P(k+1)$为真”。

I4. [k 加 1] k 加 1 并转到步骤 I2。▌

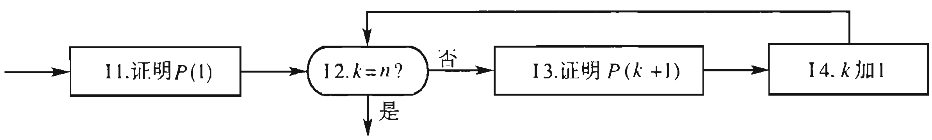


图 2　算法 I:数学归纳法

由于这个算法清楚地给出对于任意给定的 n,$P(n)$的一个证明,这个由步骤 a)和 b)组成的证明技术在逻辑上是正确的,它被叫做*用数学归纳法证明*。

数学归纳法的概念不同于通常所说的科学上的归纳推理。一个科学家进行特定的

观察,再通过"归纳法"建立起说明所观察事实的一般理论或假说;例如,我们可以观察上边(1)中的五个关系,并表达成公式(2)。在这个意义下,归纳至多是我们关于这个情况的最好的猜测;数学家称它为一个经验结果或猜想。

看看另一个例子也是有益处的。令 $p(n)$ 表示 n 的分划(partition)个数,即把 n 写成为正整数之和但不管其顺序的不同方式的数目。由于 5 正好可以有 7 种方式来分划:

$$1+1+1+1+1=2+1+1+1=2+2+1=3+1+1=3+2=4+1=5$$

我们有 $p(5)=7$。事实上,容易建立前面几个值

$$p(1)=1,\ p(2)=2,\ p(3)=3,\ p(4)=5,\ p(5)=7$$

这时,由归纳法,我们可能倾向于假设序列 $p(2),p(3),\cdots$ 取遍素数。为了检验这个假设,我们来计算 $p(6)$,看哪! $p(6)=11$! 证实了我们的猜想。

[不幸,$p(7)$ 结果是 15,坏了事了,因此我们必须再试其它假设。数 $p(n)$ 被公认是十分复杂的,尽管 S. Ramanujan 成功地猜测并证明了关于它们许多值得注意的结论。欲了解进一步的信息,请参看 G. H. Hardy,*Ramanujan* (London: Cambridge University Press, 1940),第 6 章和第 8 章。另见 7.2.1.4 小节。]

数学归纳法十分不同于刚才说明的归纳法。它不仅仅是猜测的工作,而是一个命题的结论性的证明。其实,它是无穷多个命题(对于每个 n 都有一个)的一个证明。它被叫做归纳法仅仅因为在可以应用数学归纳法的技术之前,人们首先必须判断要加以证明的是什么。今后,在本书中,仅当我们希望特指数学归纳法证明时,我们才使用归纳法一词。

有一个证明等式(2)的几何方法。例如,对于 $n=6$,图 3 示出,n^2 个方格被分成为 $1+3+\cdots+(2n-1)$ 个方格的分组。然而,在最后的分析中,仅当我们能够证明这一构造适用于所有的 n 时,这个图才能被认为是一个"证明",而且这样一个图解实际上和归纳法是一样的。

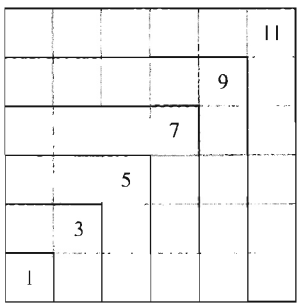


图 3 奇数之和是一个平方

我们对等式(2)的证明仅仅使用 b)的一个特殊情况,我们仅仅证明 $P(n)$ 为真意味着 $P(n+1)$ 为真。这是一个经常出现的重要的简单情况,但是下边的例子将进一步说明这个方法的威力。我们通过 $F_0=0$、$F_1=1$,以及以后的每一项都是前两项之和这样一个规则定义斐波那契(Fibonacci)序列 $F_0,F_1,F_2,\cdots$。因此这个序列由 0,1,1,2,3,5,8,13,…开始;我们将在 1.2.8 小节详细地研究它。现在将证明如果 ϕ 是数 $(1+\sqrt{5})/2$,对于所有正整数 n,我们有

$$F_n \leqslant \phi^{n-1} \tag{3}$$

称这个公式为 $P(n)$。

如果 $n=1$,则 $F_1=1=\phi^0=\phi^{n-1}$,因此步骤 a)已经完成。对于步骤 b),我们首先注意 $P(2)$ 也为真,因为 $F_2=1<1.6<\phi^1=\phi^{2-1}$。现在,如果所有 $P(1),P(2),\cdots,P(n)$ 都

为真且 $n>1$，特别是我们知道 $P(n-1)$ 和 $P(n)$ 为真，因此 $F_{n-1}\leqslant\phi^{n-2}$ 和 $F_n\leqslant\phi^{n-1}$。把这些不等式相加，我们得到

$$F_{n+1} = F_{n-1} + F_n \leqslant \phi^{n-2} + \phi^{n-1} = \phi^{n-2}(1+\phi) \tag{4}$$

数 ϕ 的重要性质，即我们首先对于这个问题选择这个数的确实原因，是

$$1 + \phi = \phi^2 \tag{5}$$

把(5)代入到(4)中给出 $F_{n+1}\leqslant\phi^n$，它就是 $P(n+1)$。所以步骤 b)已经完成，因而我们已经通过归纳法证明了(3)。注意在这里我们用两种不同的方法处理步骤 b)：当 $n=1$ 时，我们直接证明 $P(n+1)$，而当 $n>1$ 时，我们使用一个归纳方法。这是必要的，因为当 $n=1$ 时，我们对于 $P(n-1)=P(0)$ 的应用不是合法的。

数学归纳法也可用来证明关于算法的事情。考虑欧几里得算法的如下扩充：

算法 E（*扩充的欧几里得算法*）　给定两个正整数 m 和 n，我们计算它们的最大公因子 d 和两个整数 a 和 b，使得 $am+bn=d$。

E1.［初始化］置 $a'\leftarrow b\leftarrow 1, a\leftarrow b'\leftarrow 0, c\leftarrow m, d\leftarrow n$。

E2.［除］设 q 和 r 分别是 c 除以 d 的商和余数。（我们有 $c=qd+r$，且 $0\leqslant r<d$。）

E3.［余数为 0?］如果 $r=0$，算法终止；在此情况下，我们如愿地有 $am+bn=d$。

E4.［循环］置 $c\leftarrow d,\ d\leftarrow r, t\leftarrow a', a'\leftarrow a, a\leftarrow t-qa, t\leftarrow b', b'\leftarrow b, b\leftarrow t-qb$，并返回 E2。▌

如果从这个算法中删除变量 a、b, a', b'，而且使用 m 和 n 作为辅助变量 c 和 d，我们就得到原先的算法 1.1E。除了确定系数 a 和 b 外，新的算法并没有多做什么事情。假设 $m=1769$ 和 $n=551$，我们逐次地有（在步骤 E2 之后）：

a'	a	b'	b	c	d	q	r
1	0	0	1	1769	551	3	116
0	1	1	−3	551	116	4	87
1	−4	−3	13	116	87	1	29
−4	5	13	−16	87	29	3	0

答案是正确的：$5\times1769-16\times551=8845-8816=29$，即 1769 和 551 的最大公因子。

这个问题是要证明，对于所有 m 和 n 这个算法都有效。我们可以通过设 $P(n)$ 是"算法 E 对于 n 和所有整数 m 有效"这一命题，而试图应用数学归纳法。然而，那种方法并不那么容易。而且我们还需要证明某些额外的事实。在稍稍进行研究之后，我们发现必须对 a, b, a', b' 作某些证明，而适当的事实是，每当执行步骤 E2 时，等式

$$a'm + b'n = c, \qquad am + bn = d \tag{6}$$

总是成立。注意到当我们头一次到达 E2 时它们肯定为真，而步骤 E4 不改变它们的正确性，我们就可以直接证明这些等式（见习题 6）。

现在我们已经做好了通过对 n 用归纳法，证明算法 E 有效的准备。如果 m 是 n 的倍数，这个算法明显地是正确的。因为在头一次到达步骤 E3 时我们就立即完成了。当

$n=1$ 时,这个情况总出现。剩下的惟一情况是 $n>1$ 且 m 不是 n 的倍数时。在这样的情况下,在头一次执行后,算法进行到设置 $c \leftarrow n, d \leftarrow r$,而由于 $r<n$,*由归纳法我们可以假定*,d 的最终值是 n 和 r 的最大公因子。由 1.1 节给出的论证,数偶 $\{m,n\}$ 和 $\{n,r\}$ 有相同的公因子,而且特别地,它们有相同的最大公因子。因此,d 是 m 和 n 的最大公因子,而且由(6),$am+bn=d$。

上边证明中的楷体字短语,说明在归纳法的证明中经常使用的习惯语言:当在做归纳法构造的 b)部分时,不是说"我们现在将假定 $P(1), P(2), \cdots, P(n)$,并且由这个假定,我们将证明 $P(n+1)$"。我们经常只说"我们现在将证明 $P(n)$;*由归纳法我们可以假定*,每当 $1 \leqslant k<n$ 时,$P(k)$ 为真"。

如果仔细地考察上述论证并稍微改变我们的视点,我们可以想像出可适用于证明任何算法的正确性的一般方法。想法是对某个算法给出一个流程图,并且对每个箭头标以在计算经过该箭头时有关当前事态的论断。参见图 4,其中已对断言标出 *A1*,*A2*,…,*A6*。(所有这些断言都有另外的约定,即变量均为整数;但为节省篇幅起见,这个约定已被略去。)*A1* 给出当进入这个算法时的初始假定,而 *A4* 指出关于输出值 a, b 和 d 我们希望的证明结束。

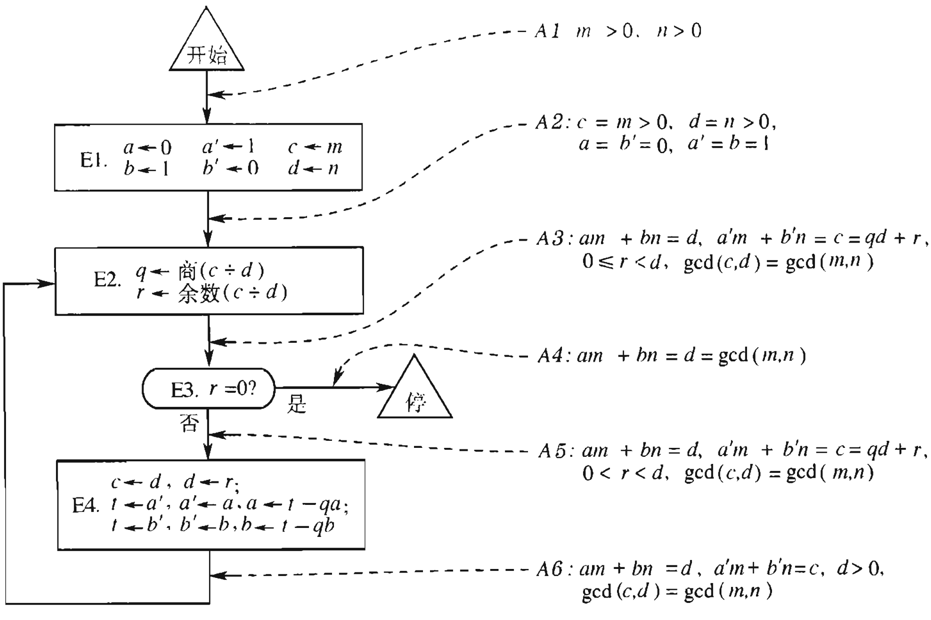


图 4 算法 E 的流程图,并标以证明算法正确性的断言

一般方法是由对于流程图中的每个框,证明下列事实组成的:

如果在执行一个框的操作之前,引向本框的任一个箭头所附的断言为真,则在该操作之后离开该框的有关箭头上的断言为真。 (7)

因此,例如,我们必须证明在 E2 之前的 *A2* 或 *A6* 蕴涵 E2 之后的 *A3*。(在此情况

下，*A2* 是比 *A6* 更强的命题；即 *A2* 蕴涵 *A6*。因此我们只需证明，在 E2 之前的 *A6* 蕴涵之后的 *A3*，注意在 *A6* 中的条件 $d>0$ 是必要的，因为它正好保证 E2 的运算有意义。）也有必要证明 *A3* 和 $r=0$ 蕴涵 *A4*；*A3* 和 $r\neq 0$ 蕴涵 *A5*，等等。每个需要的证明都是非常直截了当的。

一旦对于每个框，都证明了命题(7)，就得出在这个算法的任何执行期间，所有断言都为真。因为我们现在可以对计算的步骤数——其意义也即遍历流程图的箭头数——使用归纳法。在遍历头一个箭头，也即从“开始”引出的那个箭头时，断言 *A1* 为真，因为我们总是假定我们的输入值满足断言的规定；因此遍历头一个箭头时的断言是正确的。如果标记第 n 个箭头的断言为真，则由(7)标记第$(n+1)$个箭头的断言也为真。

使用这一般的方法，证明一个给定的算法正确的问题显然大部分是由构思出正确的断言并把它们放到流程图上组成。一旦做出了这一归纳，那么进行关于引入一个框的每个断言逻辑上蕴涵离开一个框的每个断言的证明，就只是例行公事罢了。事实上，一旦一些困难的问题已被发现，构思断言本身大半也是事务性的工作而已；因此在我们的例子中，如果已经给出了 *A1*，*A4* 和 *A6* 的话，实际上写出 *A2*，*A3* 和 *A5* 必须是什么，就非常简单了。在我们的例子中，断言 *A6* 是证明的创造性部分；所有剩下的断言原则上都可机械地加以提供。因此，对本书后面给出的算法，我们不打算像在图 4 见到的那样，给出其详细的形式证明。指出关键性的归纳法断言就足够了。这些断言或者出现在算法之后的讨论中，或者在算法本身的文字中以带括号的注释来给出。

这个证明算法正确性的方法，还有甚至更为重要的另一方面：*它反映了我们理解一个算法的途径*。回想一下，在 1.1 节中，曾经告诫读者不要期望像读一篇小说那样来读一个算法，建议读者要以某些数据来运行算法一两遍。之所以要特别如此，是因为运行算法的例子将有助于使人在头脑中形成各种断言。作者所主张的是，仅当我们达到了这样一点，即在我们的脑海中已经无保留地充满了所有断言，如同在图 4 中所完成的那样时，我们才真正理解为什么一个算法是正确的。这一观点对人们之间交流算法具有重要的心理学结果：它意味着，在向他人解释算法时，那些不能容易地通过自动化导出的关键的断言，必须明确地加以指出。当提出算法 E 时，也应提出断言 *A6*。

然而，机敏的读者将会发现在关于算法 E 的最后证明中的一个漏洞，我们根本没有证明算法终止；我们所证明的只是*如果*它终止，那它给出正确的答案！

（例如，注意到如果我们允许变量 m,n,c,d 和 r 取形如 $u+v\sqrt{2}$的值，其中 u 和 v 为整数，则算法 E 仍然有意义。变量 q,a,b,a',b' 都仍然取整数的值。如果我们以，比如说，$m=12-6\sqrt{2}$和 $n=20-10\sqrt{2}$来开始这一方法，它将计算一个最大公因子 $d=4-2\sqrt{2}$且$a=+2,b=-1$。甚至在这个扩充的假定下，断言 *A1* 到 *A6* 的证明仍然成立；因此所有断言在过程的任何执行中都为真。但是，如果我们以值 $m=1$ 和 $n=\sqrt{2}$来开始，则计算永不会终止（见习题 12）。因此断言 *A1* 到 *A6* 的证明逻辑上并不证明此算法是有穷的。）

关于终止的证明通常是分开处理的。但习题 13 证明，在许多重要的情况下有可能

来扩充上述的方法,使得关于终止的证明可作为一个副产品被包括在内。

我们现在已两次证明了算法 E 的正确性。为了做到逻辑上严格,也应当试图证明本节中的头一个算法,即算法 I 的正确性;事实上,我们已经使用算法 I 来建立任何归纳法证明的正确性。然而,如果又试图证明算法 I 本身行之有效,我们就陷入了前后矛盾的境地——如果不再次使用归纳法,我们就不能真正证明它! 这个论证将是循环的。

在最后的分析中,各整数的每一个性质也必须用归纳法大体沿着这条路线来进行证明,因为如果我们着力于研究基本概念,则整数实质上要用归纳法来定义。因此,我们可以把这样一个思想取作公理,即任何正整数或者等于 1,或者可通过由 1 开始,重复地加 1 来达到;这足以证明算法 I 是正确的。[关于整数的基础概念的严密研究,请参看 Leon Henkin 的论文"On Mathematical Induction", *AMM* **67** (1960),323 ~ 338。]

蕴涵在数学归纳法中的思想,同数的概念密切相关。最先应用数学归纳法进行严格证明的欧洲人是意大利的科学家 Francesco Maurolico,其时是 1570 年。17 世纪初,Piere de Fermat(费马)作了进一步的改进;他把它叫做"无限后继的方法"。这个思想也明显地出现于 Blaise Pascal (帕斯卡)稍后的著作中(1653)。"数学归纳法"一词是由 A. De.Morgan 在 19 世纪初创造的。[见*AMM* **24** (1917),199 ~ 207,**25** (1918),197 ~ 201; *Arch. Hist.Exac. Sci.* **9** (1972),1 ~ 21。]关于数学归纳法的进一步讨论,还可在 G. Pólya 所著的*Induction and Analogy in Mathematics* (Princeton,N.J.:Princeton University Press,1954)一书中的第 7 章找到。

如上所给出的算法证明借助于断言和归纳法的系统表述,主要应归功于 R.W. Floyd。他指出,一种程序设计语言中的每一个运算的语义定义,都可作为一个逻辑规则系统地表述。这个逻辑规则精确地指出,事先根据什么断言是正确的,在运算之后,什么断言可被证明[请参看"Assigning Meanings to Programs",*Proc. Symp. Appl. Math.*, Amer. Math. Soc., **19** (1967) 19 ~ 32]。类似的思想也由 Peter Naur 独立地提出 [*BIT* **6** (1966), 310 ~ 316],他把断言叫做"一般快照"。C. A. R. Hoare 引进了一个重要的改进,即"不变量"的概念;例如, 参看*CACM* **14** (1971),39 ~ 45。后来,一些作者发现把 Floyd 的方向颠倒过来,即从在运算之后应当成立的断言转向在运算完成之前必须成立的"最弱的先决条件",是有好处的;如果我们从所需输出的说明开始向后工作,这样一个方法使得有可能发现确保它们是正确的新算法。[参见 E. W. Dijkstra,*CACM* **18** (1975) 453 ~ 457; *A Discipline of Programming* (Prentice-Hall, 1976)。]

归纳断言的概念实际上在 1946 年就出现了雏形,与 H. H. Goldstine 和 J. von Neumann(冯诺伊曼)引进流程图同时。他们原来的流程图包括了"断言框",这些断言框非常类似于图 4 中的断言。[参见 J. von Neumam, *Collected Works* **5** (New York:Macmillan, 1963), 91 ~ 99。也可参见 A. M.Turing(图灵)早年的举证,见 *Report of a Conference on High Speed Automatic Calculating Machines* (Cambridge Univ.,1949), 67 ~ 68 和插图;该文由 F. L.Morris 和 C.B. Jones 附加评述重印于*Annals of the History of Computing* **6** (1984), 139 ~ 143。]

The understanding of the theory of a routine
may be greatly aided by providing, at the time of construction
one or two statements concerning the state of the machine
at well chosen points. . . .
In the extreme form of the theoretical method
a watertight mathematical proof is provided for the assertions.
In the extreme form of the experimental method
the routine is tried out on the machine with a variety of initial
conditions and is pronounced fit if the assertions hold in each case.
Both methods have their weaknesses.
在编写程序的时候，在合适的地方，
提供一两个涉及机器状态的命题，
可能会大大有助于对于程序理论的理解……
在理论方法一端，
对于断言应提供严密的数学证明。
在实验方法一端，
应当用各种各样的初始条件在机器上对程序进行试验，
而且如果在每种情况下断言都成立则宣告成功。
但两种方法都有其弱点。
—A. M. Turing, *Ferranti Mark I Programming Manual* (1950)

习　题

1. [*05*] 在对所有非负整数——即对于 $n = 0, 1, 2, \cdots$ 而不是 $n = 1, 2, 3, \cdots$ 证明某个命题 $P(n)$ 为真的情况下，试说明应如何修改数学归纳法证明的思想。

▶ 2. [*15*] 下列证明必定有某些错误，试指出错误之所在。“**定理**：设 a 为任意正数，对于所有正整数 n，我们有 $a^{n-1} = 1$。证明：如果 $n = 1$，$a^{n-1} = a^{1-1} = a^0 = 1$。且由归纳法，假定定理对于 $1, 2, \cdots, n$ 为真，我们有

$$a^{(n+1)-1} = a^n = \frac{a^{n-1} \times a^{n-1}}{a^{(n-1)-1}} = \frac{1 \times 1}{1} = 1$$

因此，对于 $n+1$ 定理也为真。”

3. [*18*] 以下的归纳法证明似乎是正确的，但由于某种原因，对 $n = 6$，等式的左边给出 $\frac{1}{2} + \frac{1}{6} + \frac{1}{12} + \frac{1}{20} + \frac{1}{30} = \frac{5}{6}$，而右边却给出 $\frac{3}{2} - \frac{1}{6} = \frac{4}{3}$。你能找出错误来吗？

“**定理**：

$$\frac{1}{1 \times 2} + \frac{1}{2 \times 3} + \cdots + \frac{1}{(n-1) \times n} = \frac{3}{2} - \frac{1}{n}$$

证明：对 n 用归纳法。对 $n = 1$，显然 $\frac{3}{2} - \frac{1}{n} = \frac{1}{1 \times 2}$；而且，假定对于 n，定理为真，

$$\frac{1}{1\times 2}+\cdots+\frac{1}{(n-1)\times n}+\frac{1}{n\times(n+1)}=\frac{3}{2}-\frac{1}{n}+$$
$$\frac{1}{n(n+1)}=\frac{3}{2}-\frac{1}{n}+\left(\frac{1}{n}-\frac{1}{n+1}\right)=\frac{3}{2}-\frac{1}{n+1}\text{”}$$

4. [*20*] 证明,除等式(3)外,斐波那契数满足 $F_n \geq \phi^{n-2}$。

5. [*21*] 一个素数是大于 1 的除了 1 和它本身之外无其它真因子的整数。利用这一定义以及数学归纳法,证明每个大于 1 的整数都可写成为一个或多个素数的乘积。(一个素数被认为是单个素数即它自身的“乘积”。)

6. [*20*] 证明如果在执行步骤 E4 之前,等式(6)成立,则之后它们也成立。

7. [*23*] 对于和 $1^2, 2^2-1^2, 3^2-2^2+1^2, 4^2-3^2+2^2-1^2, 5^2-4^2+3^2-2^2+1^2$等等,表达成一个公式并用归纳法证明之。

▶ 8. [*25*] (a) 用归纳法证明以下的 Nicomachus(约公元 100 年)定理:$1^3=1$, $2^3=3+5$, $3^3=7+9+11$, $4^3=13+15+17+19$,等等。(b) 利用这一结果来证明著名公式

$$1^3+2^3+\cdots+n^3=(1+2+\cdots+n)^2$$

[注:R. W. Floyd 向作者提示了这个公式的有趣的几何解释,如图 5 所示。这个思想和 Nicomachus 定理以及图 3 有关。其它的“图解”证明可在下述书中找到:Martin Gardner, *Knotted Doughnuts* (New York: Freeman, 1986),第 16 章;J. H. Conway 和 R. K. Guy, *The Book of Numbers* (New York: Copernicus, 1996),第 2 章。]

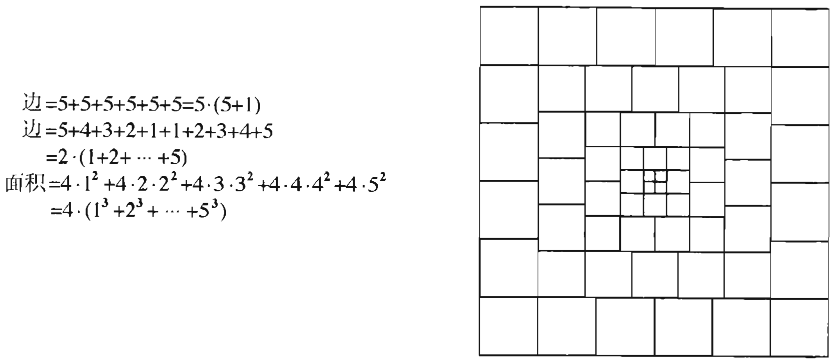


图 5 习题 8(b)的几何形式

9. [*20*] 用归纳法证明,如果 $0<a<1$,则 $(1-a)^n \geq 1-na$。

10. [*M22*] 用归纳法证明,如果 $n\geq 10$,则 $2^n > n^3$。

11. [*M30*] 求出和$\frac{1^3}{1^4+4}-\frac{3^3}{3^4+4}+\frac{5^3}{5^4+4}-\cdots+\frac{(-1)^n(2n+1)^3}{(2n+1)^4+4}$的简单公式并证明之。

12. [*M25*] 试说明可以怎样推广算法 E,使得如正文中所述那样,它将接受形如 $u+v\sqrt{2}$的输入值,其中 u 和 v 都是整数,而且计算仍能以初等的方式进行(即无须使用$\sqrt{2}$的无限的小数展开)。然而,试证明,如果 $m=1$ 和 $n=\sqrt{2}$,则计算将不终止。

▶ 13. [*M23*] 通过加上一个新变量 T 和在每步的开始处加上“$T\leftarrow T+1$”的运算来扩充算法 E。(这样,T 就好像是个时钟一样,计算着执行的步骤数)。假设 T 开始为 0,故图 4 中的断言 *A1* 变成“$m>0, n>0, T=0$”。附加的条件“$T=1$” 将类似地添加到 *A2* 上。说明如何以这样的方式把

附加的条件附加到断言中，使得 A1，A2，…，A6 中的任何一个都蕴涵 $T\leqslant 3n$，而且使得归纳法证明仍然得以进行。（因此计算必然在至多 $3n$ 步内终止。）

14. [*50*]（R.W.Floyd）试编制一个计算机程序，它接受某种程序设计语言的程序连同任选的断言一起作为输入，而且它试图添进剩下的为证明此计算机程序正确的断言。（例如，努力编出一个能证明算法 E 的正确性的程序，假定仅仅给出断言 A1，A4 和 A6。关于这个问题的进一步讨论，请参看 R. W. Floyd 和 J. C. King 的论文，发表在 1971 年的国际信息处理联合会大会（IFIP Congress）论文集上。）

▶ 15. [*HM28*]（*广义归纳法*）正文中说明了怎样证明依赖于一个整数 n 的命题 $P(n)$，但它没有描述怎样证明依赖于两个整数的命题 $P(m,n)$。在这些情况下，通常是通过某种类型的"双重归纳法"来给出一个证明的，因而经常显得含糊不清。实际上，有一个比简单的归纳法更为一般的重要原理，它不仅适用于这种情况，而且还适用于关于不可数集合证明命题的情况——例如，对于所有实数 x 的 $P(x)$。这个一般原理叫做*良序原理*（well-ordering）。

设"$\prec$"是集合 S 上的一个关系，它满足以下的性质：

i) 给定 S 中的 x，y 和 z，如果 $x\prec y$ 和 $y\prec z$，则 $x\prec z$。

ii) 给定 S 中的 x 和 y，以下三种可能性中恰有一种为真：$x\prec y$，$x=y$ 或 $y\prec x$。

iii) 如果 A 是 S 的任何非空子集，则 A 中有一个元素 x，使得对于 A 中所有的 y，有 $x\preceq y$（即 $x\prec y$ 或 $x=y$）。

这个关系被称为 S 的一个良序关系。例如，对于通常的"小于"关系 $<$，正整数显然是良序的。

a) 试证明所有整数的集合对于 $<$ 不是良序的。

b) 对于所有整数的集合，试定义一个良序关系。

c) 所有非负实数的集合对于 $<$ 是否良序？

d)（字典序）设 S 对于 $\prec$ 良序，而且对于 $n>0$ 令 T_n 是 S 中元素 x_j 的所有 n 元组 $(x_1,x_2,\cdots,x_n)$ 的集合。如果有某一个 k，$1\leqslant k\leqslant n$，使得在 S 中，对于 $1\leqslant j<k$，$x_j=y_j$，但是 $x_k\prec y_k$，则定义 $(x_1,x_2,\cdots,x_n)\prec(y_1,y_2,\cdots,y_n)$。问 $\prec$ 是否 T_n 的一个良序？

e) 继续小题 d)，令 $T=\bigcup_{n\geqslant 1}T_n$；定义 $(x_1,x_2,\cdots,x_n)\prec(y_1,y_2,\cdots,y_n)$，如果对于某 $k\leqslant\min(m,n)$，当 $1\leqslant j<k$ 时，$x_j=y_j$，且 $x_k\prec y_k$，或者如果 $m<n$，且对于 $1\leqslant j\leqslant m$，$x_j=y_j$。$\prec$ 是否 T 的一个良序？

f) 试证 $\prec$ 是 S 的一个良序当且仅当它满足上边的 i) 和 ii)，而且对于所有 $j\geqslant 1$，不存在具有 $x_{j+1}\prec x_j$ 的无穷序列 $x_1,x_2,x_3,\cdots$。

g) 设 S 对于 $\prec$ 是良序，且设 $P(x)$ 是关于 S 的元素 x 的一个命题。试证如果在对于所有的 $y\prec x$，$P(y)$ 为真的假定下能证得 $P(x)$ 为真，则 $P(x)$ 对 S 中的所有 x 为真。

[注：小题 g) 是原来约定的简单归纳法的推广；在 S = 正整数的情况下，它只不过是正文中讨论的数学归纳法的简单情况。在该情况下，要求我们证明，对于所有正整数 $y<1$，如果 $P(y)$ 为真，则 $P(1)$ 为真；这就等于说，我们应当证明 $P(1)$，因为对于所有这样的 y，$P(y)$ 肯定（空地）为真。结果，人们发现，在许多情况下，使用一个特殊的论证，$P(1)$ 就不必证明了。

小题 d) 同小题 g) 相联系，特别为我们提供了用于证明关于 n 个正整数 $m_1,\cdots,m_n$ 的命题 $P(m_1,\cdots,m_n)$ 的 n 元组归纳法的更强有力的方法。

小题 f) 对于计算机算法有着进一步的应用：如果我们能把一个计算的诸状态 x 映射到属于良序集 S 的一个元素 $f(x)$，且使得计算的每一步把一个状态 x 转换成状态 y，并有 $f(y)\prec f(x)$，

则算法必然终止。这一原理推广了在证明算法 1.1E 终止中所使用的关于 n 的值严格递降的论证。]

1.2.2 数,幂和对数

现在让我们好好考察我们一直都在讨论的数,以开始我们对于数值数学的研究。整数是如下全体数(负的,零和正的):

$$\cdots, -3, -2, -1, 0, 1, 2, 3, \cdots$$

有理数是两个整数的比(商)p/q,其中 q 为正。实数是有一个十进展开的量 x:

$$x = n + 0.d_1 d_2 d_3 \cdots \tag{1}$$

其中 n 是一个整数,每个 d_i 是 0 和 9 之间的一个数字,而且这个数字序列不以无穷多个 9 结尾。(1)的表示意味着对于所有正整数 k,

$$n + \frac{d_1}{10} + \frac{d_2}{100} + \cdots + \frac{d_k}{10^k} \leqslant x < n + \frac{d_1}{10} + \frac{d_2}{100} + \cdots + \frac{d_k}{10^k} + \frac{1}{10^k} \tag{2}$$

不是有理数的实数的例子有:

$\pi = 3.14159265358979\cdots$,在一个圆中圆周同直径之比;

$\phi = 1.61803398874989\cdots$,黄金比率$(1+\sqrt{5})/2$(见 1.2.8 小节)。

附录 A 中给出了有 40 位小数精确度的重要常数表。我们不需讨论关于实数的加法,减法,乘法以及比较等熟知的性质。

关于整数的难题通常是通过使用实数来解决的。而关于实数的难题通常是通过使用更一般的称做复数的一类值而解决的。复数是形如 $z = x + iy$ 的量 z,其中 x 和 y 是实数,而 i 是满足等式 $i^2 = -1$ 的一个特殊的量。我们称 x 和 y 为 z 的实部和虚部,并定义 z 的绝对值为:

$$|z| = \sqrt{x^2 + y^2} \tag{3}$$

z 的复共轭是$\bar{z} = x - iy$,而且我们有 $z\bar{z} = x^2 + y^2 = |z|^2$。复数理论在许多方面比实数理论更简单和更优美,但通常把它看做高等的课题。因此我们在本书中将只专注于实数,除非当实数变得不必要地复杂时。

如果 u 和 v 是满足 $u \leqslant v$ 的实数,闭区间$[u,v]$是使得 $u \leqslant x \leqslant v$ 的实数 x 的集合。类似地,开区间(u,v)是使得 $u < x < v$ 的 x 的集合。半开区间$[u,v)$或$(u,v]$以类似方式定义。我们也允许在一个开端点处 u 为 $-\infty$ 或 v 为 ∞,表示无下界或无上界;因此$(-\infty,\infty)$代表所有实数的集合。而$[0,\infty)$表示非负实数。

贯穿于这一小节,令字母 b 代表一个正实数。如果 n 是一个整数,则 b^n 以熟知的规则定义:

$$b^0 = 1, b^n = b^{n-1}b \text{ 如果 } n > 0, \quad b^n = b^{n+1}/b \text{ 如果 } n < 0 \tag{4}$$

通过归纳法容易证明当 x 和 y 为整数时,指数律成立:

$$b^{x+y}=b^x b^y,\qquad (b^x)^y=b^{xy} \tag{5}$$

如果 u 是一个正实数，m 是一个正整数，则总存在惟一的正实数 v 使得 $v^m=u$；它称做 u 的 m 次根，并记作 $v=\sqrt[m]{u}$。

我们现在对于有理数 $r=p/q$ 定义 b^r 如下：

$$b^{p/q}=\sqrt[q]{b^p} \tag{6}$$

Oresme(大约 1360 年)提出的这一定义是一个很好的定义，因为 $b^{ap/aq}=b^{p/q}$，而且即使当 x 和 y 是任意的有理数时指数律仍然成立(见习题 9)。

最后，对于所有实数值 x，我们定义 b^x。首先假设 $b>1$；如果 x 由式(1)给出，我们要求

$$b^{n+d_1/10+\cdots+d_k/10^k}\leqslant b^x<b^{n+d_1/10+\cdots+d_k/10^k+1/10^k} \tag{7}$$

这就把 b^x 定义为惟一的正实数，因为在式(7)中右端和左端间的差是 $b^{n+d_1/10+\cdots+d_k/10^k}\cdot(b^{1/10^k}-1)$；由以下的习题(13)，这个差小于 $b^{n+1}(b-1)/10^k$，而且如果我们把 k 取得充分大，对于 b^x 我们可以由此得到任何希望的精度。

例如，我们发现

$$10^{0.30102999}=1.9999999739\cdots,\qquad 10^{0.30103000}=2.0000000199\cdots \tag{8}$$

因此，如果 $b=10$ 和 $x=0.30102999\cdots$，我们就知道 10^x 的精度优于千万分之一(尽管我们仍然不知道 10^x 的展开是 $1.999\cdots$ 还是 $2.000\cdots$)。

当 $b<1$ 时，我们定义 $b^x=(1/b)^{-x}$；而且当 $b=1$ 时，$b^x=1$。通过这些定义，可以证明，对于 x 和 y 的任何实数值，指数律(5)成立。定义 b^x 的这些思想是由 John Wallis (1665)和 Isaac Newton(1669)首先系统表述的。

现在我们转到一个重要的问题。假设给定一个正的实数 y，我们能否找到一个实数 x 使得 $y=b^x$？回答是“能”(只要是 $b\neq1$)，因为当给定 $b^x=y$ 时，我们可以简单地反过来使用等式(7)来确定 n 和 $d_1,d_2,\cdots$。得到的数 x 叫做 y 的对于底 b 的对数。我们把这写成 $x=\log_b y$。根据这个定义，我们有

$$x=b^{\log_b x}=\log_b(b^x) \tag{9}$$

作为一个例子，等式(8)表明

$$\log_{10}2=0.30102999\cdots \tag{10}$$

由指数律得出

$$\log_b(xy)=\log_b x+\log_b y,\quad 如果\ x>0, y>0 \tag{11}$$

且

$$\log_b(c^y)=y\log_b c,\quad 如果\ c>0 \tag{12}$$

等式(10)说明所谓的*常用对数*,即底是10时的对数。人们可能会想到,在计算机的工作中,*二进制对数*(以2为底)将是更有用的,因为大多数的计算机都采用二进制的算术。实际上,我们将看到,二进制对数确实非常有用,但并非仅仅由于这个原因;主要原因在于,一个计算机算法,通常都发生两路分支。二进制对数如此经常地出现,因而对于它们有一个短一些的记号是明智的,因此我们将遵循 Edward M. Reingold 的提议,写成

$$\lg x = \log_2 x \tag{13}$$

现在又提出在 $\lg x$ 和 $\log_{10} x$ 之间是否有任何关系的问题。很幸运,确实有,由等式(9)和(12),

$$\log_{10} x = \log_{10}(2^{\lg x}) = (\lg x)(\log_{10} 2)$$

因此 $\lg x = \log_{10} x / \log_{10} 2$,而且一般地,我们有

$$\log_c x = \frac{\log_b x}{\log_b c} \tag{14}$$

等式(11),(12)和(14)是进行对数运算的基本规则。

在大多数情况下,底10或底2都不是真正最方便的用于计算的底。有一个以 $e = 2.718281828459045\cdots$ 表示的实数,对于它对数有较为简单的性质。以e为底的对数被方便地叫做*自然对数*,我们写

$$\ln x = \log_e x \tag{15}$$

这一稍微随意的定义(事实上我们还并未真正地定义 e),大概不能使读者信服是一个非常“自然”的对数。但我们将会发现,我们越是对它使用得多,就会越觉得 $\ln x$ 自然。John Napier 实际上在公元1590年之前就发现了自然对数(稍微有些变化,而且没有把它们同乘幂联系起来),这比知道其它类型的对数要早许多年。下面给出的两个例子,是在每本微积分教科书中都要证明的,为我们提供了为什么 Napier 的对数值称做“自然”的一些启示:(a)在图6中,带阴影部分的面积是 $\ln x$。(b)如果一个银行以复利率 r 按半年支付复利,每1元钱一年的本息为 $(1 + r/2)^2$ 元;如果以季来计算复利,你得到的本息将是 $(1 + r/4)^4$ 元。因此如果是按天计算复利,你大概会得到 $(1 + r/365)^{365}$ 元。现在如果利息是*连续地*按复利计,则对于每一元你将恰好得到 e^r 元(忽略舍入误差)。在现今这个计算机时代里,许多银行家实际上已经达到这个极限公式。

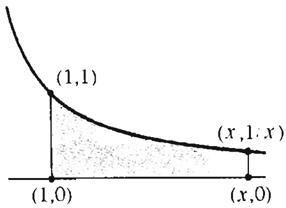


图6 自然对数

关于对数和指数概念的有趣的历史,请参看 F. Cajori 的一系列论文:*AMM* **20** (1913),5~14,35~47,75~84,107~117,148~151,173~182,205~210。

我们以考虑如何计算对数来结束这一小节。一个方法是由等式(7)直接提示的:如果令 $b^x = y$,并且把等式(7)的所有部分都做 10^k 次方,我们发现,对于某个整数 m,

$$b^m \leqslant y^{10^k} < b^{m+1} \tag{16}$$

为了得到 y 的对数,我们所要做的就是把 y 做这么大的乘方并找出结果落在 b 的哪一个次幂$(m,m+1)$上;然后 $m/10^k$ 是精确到 k 个小数位置的答案。

对这个明显的不实用的方法稍做修改,就导致了一个简单和合理的步骤。我们现在就来说明怎样计算 $\log_{10} x$,并且以二进制系统来表达答案:

$$\log_{10} x = n + b_1/2 + b_2/4 + b_3/8 + \cdots \tag{17}$$

首先我们把 x 的小数点向左移或向右移使得 $1 \leqslant x/10^n < 10$;这就确定了整数部分 n。为了得到 $b_1, b_2, \cdots$,我们现在置 $x_0 = x/10^n$,而且对于 $k \geqslant 1$,

$$\begin{aligned} b_k &= 0, x_k = x_{k-1}^2, &\quad \text{如果 } x_{k-1}^2 < 10 \\ b_k &= 1, x_k = x_{k-1}^2/10, &\quad \text{如果 } x_{k-1}^2 \geqslant 10 \end{aligned} \tag{18}$$

这一步骤的正确性从下列事实得出,对于 $k = 0, 1, 2, \cdots$,

$$1 \leqslant x_k = x^{2^k}/10^{2^k(n+b_1/2+\cdots+b_k/2^k)} < 10 \tag{19}$$

而这是容易用归纳法证明的。

当然,在实践中,我们必然只能以有限的精度进行计算,所以我们不能精确地置 $x_k = x_{k-1}^2$。代替的是,我们置 $x_k = x_{k-1}^2$舍入或截断到某位小数。例如,下面是对 $\log_{10} 2$ 舍入到 4 位有效数字的计算:

$x_0 = 2.000$;
$x_1 = 4.000$,　$b_1 = 0$;　　$x_6 = 1.845$,　$b_6 = 1$;
$x_2 = 1.600$,　$b_2 = 1$;　　$x_7 = 3.404$,　$b_7 = 0$;
$x_3 = 2.560$,　$b_3 = 0$;　　$x_8 = 1.159$,　$b_8 = 1$;
$x_4 = 6.554$,　$b_4 = 0$;　　$x_9 = 1.343$,　$b_9 = 0$;
$x_5 = 4.295$,　$b_5 = 1$;　　$x_{10} = 1.804$,　$b_{10} = 0$;　等等。

计算误差已经引起错误的传播;x_{10}真正的舍入值是 1.798,这将最终导致 b_{19}不能正确地被计算,而且我们得到二进制值$(0.0100110100010000011\cdots)_2$,它对应于十进的等价值 0.301031…而不是在等式(10)中给出的真正的值。

对任何像这样的方法,都有必要考察由于强加的限制所引起的计算误差的大小。习题 27 推导了这一误差的上界。如同上边一样,计算到 4 位数,我们发现,对数值的误差保证小于 0.00044。我们上边的答案比这还要精确,主要是因为 x_0, x_1, x_2 和 x_3 是精确地得到的。

这个方法简单而十分有趣,但大概不是在计算机上计算对数的最好方法,习题 25 给出了另一个方法。

习　题

1. [*00*] 什么是最小的正有理数?

2. [*00*] $1+0.239999999\cdots$是一个十进展开吗?

3. [*02*] $(-3)^{-3}$等于什么?

▶ 4. [*05*] $(0.125)^{-2/3}$等于什么?

5. [*05*] 我们是借助于十进展开来定义实数的。试讨论如何另外用二进展开来定义它并且给出代替等式(2)的一个定义。

6. [*10*] 设 $x=m+0.d_1d_2\cdots$ 和 $y=n+0.e_1e_2\cdots$都是实数。基于十进表示,给出确定是否 $x=y, x<y$ 或 $x>y$ 的一个规则。

7. [*M23*] 给定 x 和 y 都是整数,从等式(4)给出的定义开始,试证明指数律。

8. [*25*] 设 m 是一个正整数。通过给出逐次地构造根的十进展开中的值 $n, d_1, d_2, \cdots$,证明每一个正实数 u 都有惟一的一个 m 次根。

9. [*M23*] 给定 x 和 y 为有理数,在对于 x 和 y 为整数时指数律成立的假定下,试证明对于有理数的指数律。

10. [*18*] 试证明 $\log_{10}2$ 不是一个有理数。

▶ 11. [*10*] 如果 $b=10$ 和 $x\approx\log_{10}2$,为了确定 b^x 的十进展开的头三位小数,我们需要知道多少位精度的 x 的值?(注:在你的讨论中可以使用习题 10 的结果。)

12. [*02*] 说明为什么等式(10)是由等式(8)推出的?

▶ 13. [*M23*] (a)给定 x 是一个正实数和 n 是一个正整数,试证不等式 $\sqrt[n]{1+x}-1\le x/n$。(b)利用这一事实来验证(7)之后的说明。

14. [*15*] 证明等式(12)。

15. [*10*] 证明或否定:如果 $x, y>0$,则

$$\log_b x/y = \log_b x - \log_b y$$

16. [*00*] 如何用 $\ln x$ 和 $\ln 10$ 表达 $\log_{10}x$?

▶ 17. [*05*] $\lg 32$ 等于多少? $\log_\pi \pi$ 呢? $\ln e$ 呢? $\log_b 1$ 呢? $\log_b(-1)$呢?

18. [*10*] 证明或否定:$\log_8 x = \frac{1}{2}\lg x$。

▶ 19. [*20*] 如果 n 是十进表示有 14 位数长的一个整数,试问 n 的值能否装入有 47 个二进位和一个符号位的计算机字内?

20. [*10*] 在 $\log_{10}2$ 和 $\log_2 10$ 之间是否有什么简单关系?

21. [*15*](对数的对数)借助于 $\ln\ln x, \ln\ln b$ 和 $\ln b$ 来表达 $\log_b\log_b x$。

▶ 22. [*20*](R.W.Hamming) 试证明　$\log_{10}2+\ln 2 \doteq 0.9942$

$$\lg x \approx \ln x + \log_{10} x$$

且误差小于 1%!(因此也可用自然对数表和常用对数表求二进对数的近似值。)

23. [*M25*] 基于图 6,给出 $\ln xy=\ln x+\ln y$ 的一个几何证明。

24. [*15*] 试说明在本小节末尾中用于计算以 10 为底的对数的方法可以如何修改,以产生以 2 为底的对数。10是不好找,换成2即可.

25. [*22*] 假设我们有一台二进制的计算机和一个数 $x, 1\le x<2$。说明以下算法,它只使用和所要求的精确位数成比例的移位、加法和减法操作,可以用来计算 $y=\log_b x$ 的近似值。

L1.[初始化] 置 $y\leftarrow 0, z\leftarrow x$ 右移 1 位,$k\leftarrow 1$。

L2.[测试结束否] 如果 $x=1$,则停止。

L3.[比较] 如果 $x-z<1$,则置 $z\leftarrow z$ 右移 1 位,$k\leftarrow k+1$,并重复这一步骤。

L4.[减少值] 置 $x \leftarrow x - z$, $z \leftarrow x$ 右移 k 位, $y \leftarrow y + \log_b(2^k/(2^k - 1))$,并转到 L2。▌

[注:这个方法非常类似于在计算机硬件中做除法的方法。其想法实质上可追溯到 Henry Briggs,他使用它(以十进制而不是二进制)来计算对数表,发表于 1624 年。我们需要有和计算机的精度同样多位的常数 $\log_b 2$, $\log_b(4/3)$, $\log_b(8/7)$ 等等的一个辅助表。这个算法涉及当数向右移时的人为计算误差,使得 x 最终将被减小到 1,而且算法将终止。本题的目的是说明为什么它将终止和为什么它计算出 $\log_b x$ 的一个近似值。]

26.[*M27*] 基于在算术运算中所使用的精度,试找出上一道题中算法误差的严格上界。

▶ 27.[*M25*] 考虑正文中讨论的用于计算 $\log_{10} x$ 的方法。令 x'_k 表示所计算的 x_k 的近似值,并确定如下:$x(1-\delta) \le 10^n x'_0 \le x(1+\epsilon)$;而且在通过等式(18)确定 x'_k 当中,量 y_k 用来代替 $(x'_{k-1})^2$,其中 $(x'_{k-1})^2(1-\delta) \le y_k \le (x'_{k-1})^2(1+\epsilon)$,以及 $1 \le y_k < 100$。这里 δ 和 ϵ 都是小常数,用来反映舍入或截断引起的误差的上界和下界。如果 $\log' x$ 表示计算的结果,试证明在 k 步之后,我们有

$$\log_{10} x + 2\log_{10}(1-\delta) - 1/2^k < \log' x \le \log_{10} x + 2\log_{10}(1+\epsilon)$$

28.[*M30*](R. Feynman) 仅使用移位、加法和减法(类似于习题 25 中的算法),建立计算 $0 \le x < 1$ 时的 b^x 的值的一个算法,并分析其精度。

29.[*HM20*] 设 x 是大于 1 的一个实数。(a)对于什么样的实数 $b > 1$, $b \log_b x$ 取极小值?(b)对于什么整数 $b > 1$, $(b+1)\log_b x$ 取极小值?(c)对于什么整数 $b > 1$, $(b+1)\log_b x$ 取极小值?

1.2.3 和与积

设 $a_1, a_2, \cdots$ 是数的任意序列。我们经常对诸如 $a_1 + a_2 + \cdots + a_n$ 这样的和感兴趣,而且用下边两个等价符号中的任何一个把这个和可写得更紧凑:

$$\sum_{j=1}^{n} a_j \qquad 或 \qquad \sum_{1 \le j \le n} a_j \tag{1}$$

如果 n 为 0 或负数,则 $a_1 + a_2 + \cdots + a_n = \sum_{j=1}^{n} a_j = \sum_{1 \le j \le n} a_j$ 的值定义作 0。一般地说,如果 $R(j)$ 是关于 j 的任意关系,记号

$$\sum_{R(j)} a_j \tag{2}$$

表示所有 a_j 之和,其中 j 是满足条件 $R(j)$ 的整数。如果不存在这样的 j,则记号(2)表示 0。(1)和(2)中的字母 j 只是为上述表示法引进的**哑下标**或**下标变量**。用作下标变量的符号通常是字母 i, j, k, m, n, r, s, t(偶尔还带有下标或撇号)。在(1)和(2)中所用的大求和号也可更紧凑地写成 $\sum_{j=1}^{n} a_j$ 或 $\sum_{R(j)} a_j$。使用 $\sum$ 和下标变量来表示有确定极限的求和是由 *J. Fourier*(傅里叶)于 1820 年引进的。

严格地说,记法 $\sum_{1 \le j \le n} a_j$ 是含混的,因为它并未明确地指出求和是对于 j 还是对于 n 进行的。在这个具体的情况下,把它解释成对于 $n \ge j$ 的值的求和,是不明智的;但是使用未明确说明的下标变量却构造出有意义的例子,如同在 $\sum_{j \le k} \binom{j+k}{2j-k}$ 中这样。在这样的情况下,必须交待清楚哪一个变量是哑变量,而哪一个变量还要在这个记

法之外出现，还有另外的意义。像 $\sum_{j\leqslant k}\binom{j+k}{2j-k}$ 这样的和大概仅当 j 或者 k（但不是两者都是）有外部意义时才使用。

在大多数情况下，仅当和数是有限时，即仅当有限个 j 值满足 $R(j)$，且 $a_j\neq 0$ 时，我们才使用符号(2)。如果要求一个无穷的和，例如

$$\sum_{j=1}^{\infty} a_j = \sum_{j\geqslant 1} a_j = a_1 + a_2 + a_3 + \cdots$$

且有无穷多个非零的项，则必须使用微积分的技术；(2)的精确意义就成为

$$\sum_{R(j)} a_j = \Big(\lim_{n\to\infty} \sum_{\substack{R(j)\\ -n\leqslant j<0}} a_j\Big) + \Big(\lim_{n\to\infty} \sum_{\substack{R(j)\\ 0\leqslant j<n}} a_j\Big) \tag{3}$$

假定两个极限都存在。如果有一个或两个极限不存在，则无限和是发散的；它不存在。否则它是收敛的。

当在$\sum$符号下放置两个或多个条件时，像在(3)中那样，则表示所有条件都必须成立。

关于求和，有四个简单的代数运算非常重要，而且只有熟悉它们之后才有可能求解许多问题。我们现在就来讨论这四个运算。

a) 对于和数之积的分配律。

$$\Big(\sum_{R(i)} a_i\Big)\Big(\sum_{S(j)} b_j\Big) = \sum_{R(i)}\Big(\sum_{S(j)} a_i b_j\Big) \tag{4}$$

为了理解这个定律，可考虑特殊情况，例如，

$$\Big(\sum_{i=1}^{2} a_i\Big)\Big(\sum_{j=1}^{3} b_j\Big) = (a_1 + a_2)(b_1 + b_2 + b_3) =$$

$$(a_1 b_1 + a_1 b_2 + a_1 b_3) + (a_2 b_1 + a_2 b_2 + a_2 b_3) =$$

$$\sum_{i=1}^{2}\Big(\sum_{j=1}^{3} a_i b_j\Big)$$

习惯上通常把(4)右边的括号去掉；一个双重求和如 $\sum_{R(i)}(\sum_{S(j)} a_{ij})$ 就简单地写成 $\sum_{R(i)}\sum_{S(j)} a_{ij}$。

b) 改变变量：

$$\sum_{R(i)} a_i = \sum_{R(j)} a_j = \sum_{R(p(j))} a_{p(j)} \tag{5}$$

这个等式表示了两种类型的变换。在头一种情况下，我们只不过是把下标变量的名称从 i 改成 j。第二种情况更有意思，这里 $p(j)$ 是 j 的函数，它表示相关值的一个排列(permutation)；更精确地说，对于每个满足关系 $R(i)$ 的整数 i，必定恰好存在满足关系 $p(j)=i$ 的一个整数 j。在 $p(j)=c+j$ 和 $p(j)=c-j$ 的重要情况下（其中 c 是不依赖于 j 的一个整数），这个条件总是满足的，而且这两者都是在应用中最常使用的情况。例如，

$$\sum_{1\leqslant j\leqslant n} a_j = \sum_{1\leqslant j-1\leqslant n} a_{j-1} = \sum_{2\leqslant j\leqslant n+1} a_{j-1} \tag{6}$$

读者应当仔细地研究这一例子。

对于所有的无限求和，不能以 $p(j)$ 来代替 j。像上边那样，如果 $p(j)=c\pm j$，这一运算总是正确的，但在其它情况下，则必须小心从事。[例如，参见 T. M. Apostol, *Mathematical Analysis* (Reading, Mass.: Addison-Wesley, 1957)，第 12 章。为保证(5)对于整数的任何排列 $p(j)$ 的正确性，一个充分条件是 $\sum_{R(j)}|a_j|$ 存在。]

c) 交换求和的次序：

$$\sum_{R(i)}\sum_{S(j)} a_{ij} = \sum_{S(j)}\sum_{R(i)} a_{ij} \tag{7}$$

让我们考虑这个等式的一个非常简单的情况：

$$\sum_{R(i)}\sum_{j=1}^{2} a_{ij} = \sum_{R(i)}(a_{i1}+a_{i2})$$

$$\sum_{j=1}^{2}\sum_{R(i)} a_{ij} = \sum_{R(i)} a_{i1} + \sum_{R(i)} a_{i2}$$

由等式(7)，这两者是相等的。这无非是说

$$\sum_{R(i)}(b_i+c_i) = \sum_{R(i)} b_i + \sum_{R(i)} c_i \tag{8}$$

这里我们设 $b_i=a_{i1}$ 和 $c_i=a_{i2}$。

交换求和次序的运算是极其有用的，因为经常会发生这样一种情况，就是我们只知道 $\sum_{R(i)} a_{ij}$ 的一个简单形式，但却不知道 $\sum_{S(j)} a_{ij}$ 的形式。在更为一般的情况下，即关系 $S(j)$ 既依赖于 j 也依赖于 i，我们也经常需要交换求和的次序。在这种情况下，我们可以用“$S(i,j)$”来表示这个关系。求和的变换，至少在理论上，总可以如下来进行：

$$\sum_{R(i)}\sum_{S(i,j)} a_{ij} = \sum_{S'(j)}\sum_{R'(i,j)} a_{ij} \tag{9}$$

其中 $S'(j)$ 是关系“有一个整数 i 使得 $R(i)$ 和 $S(i,j)$ 两者都为真”；而 $R'(i,j)$ 是关系“$R(i)$ 和 $S(i,j)$ 两者都为真”。例如，如果求和是 $\sum_{i=1}^{n}\sum_{j=1}^{i} a_{ij}$，则 $S'(j)$ 是关系“存在一个整数 i 使得 $1\leqslant i\leqslant n$ 和 $1\leqslant j\leqslant i$”，即 $1\leqslant j\leqslant n$；而 $R'(i,j)$ 是关系“$1\leqslant i\leqslant n$ 和 $1\leqslant j\leqslant i$”，即 $j\leqslant i\leqslant n$。因此

$$\sum_{i=1}^{n}\sum_{j=1}^{i} a_{ij} = \sum_{j=1}^{n}\sum_{i=j}^{n} a_{ij} \tag{10}$$

[注，像在情况 b)中一样，交换求和次序的运算对于无穷级数来说并不总是正确的。如果这个级数是绝对收敛的，即如果 $\sum_{R(i)}\sum_{S(j)}|a_{ij}|$ 存在，则可以证明等式(7)和(9)是正确的。而且，如果 $R(i)$ 或 $S(j)$ 两者中有一个确定等式(7)中的有限和，而且出现的每个无穷和收敛，则交换是正确的。特别地，对于收敛的无穷和，等式(8)总是正确的。]

d) 处理作用域。如果 $R(j)$和 $S(j)$是任意的关系,我们有

$$\sum_{R(j)} a_j + \sum_{S(j)} a_j = \sum_{R(j)\text{或}S(j)} a_j + \sum_{R(j)\text{且}S(j)} a_j \tag{11}$$

例如,假定 $1\leqslant m\leqslant n$,则

$$\sum_{1\leqslant j\leqslant m} a_j + \sum_{m\leqslant j\leqslant n} a_j = (\sum_{1\leqslant j\leqslant n} a_j) + a_m \tag{12}$$

在这种情况下,“$R(j)$且 $S(j)$”简单地变成“$j=m$”,所以我们把第二个求和简化为“a_m”。在等式(11)的大多数应用当中,或者 $R(j)$与 $S(j)$仅对一个或两个 j 值同时满足,或者不可能有使 $R(j)$和 $S(j)$都成立的同一个 j 值。在后一种情况下,等式(11)右边的第二个求和就干脆不出现了。

现在,我们已经看到了四个处理求和的基本规则。下面,让我们更进一步说明怎样应用这些技术。

例 1

$$\sum_{0\leqslant j\leqslant n} a_j = \sum_{\substack{0\leqslant j\leqslant n\\ j\text{偶}}} a_j + \sum_{\substack{0\leqslant j\leqslant n\\ j\text{奇}}} a_j = \qquad \text{由规则 d)}$$

$$\sum_{\substack{0\leqslant 2j\leqslant n\\ 2j\text{偶}}} a_{2j} + \sum_{\substack{0\leqslant 2j+1\leqslant n\\ 2j+1\text{奇}}} a_{2j+1} = \qquad \text{由规则 b)}$$

$$\sum_{0\leqslant j\leqslant n/2} a_{2j} + \sum_{0\leqslant j\leqslant n/2} a_{2j+1}$$

这最后一步只不过是简化了$\sum$之下的关系。

例 2 设

$$S_1 = \sum_{i=0}^{n}\sum_{j=0}^{i} a_i a_j = \sum_{j=0}^{n}\sum_{i=j}^{n} a_i a_j = \qquad \text{由规则 c)[参看等式(10)]}$$

$$\sum_{i=0}^{n}\sum_{j=i}^{n} a_i a_j \qquad \text{由规则 b)}$$

只要交换 i 和 j 的名称,并且认清 $a_j a_i = a_i a_j$。如果我们以 S_2 来表示后一个求和,就有

$$2S_1 = S_1 + S_2 = \sum_{i=0}^{n}(\sum_{j=0}^{i} a_i a_j + \sum_{j=i}^{n} a_i a_j) = \qquad \text{由等式(8)}$$

$$\sum_{i=0}^{n}((\sum_{j=0}^{n} a_i a_j) + a_i a_i) = \qquad \text{由规则 d)[参看等式(12)]}$$

$$\sum_{i=0}^{n}\sum_{j=0}^{n} a_i a_j + \sum_{i=0}^{n} a_i a_i = \qquad \text{由等式(8)}$$

$$(\sum_{i=0}^{n} a_i)(\sum_{j=0}^{n} a_j) + (\sum_{i=0}^{n} a_i^2) = \qquad \text{由规则 a)}$$

$$(\sum_{i=0}^{n} a_i)^2 + (\sum_{i=0}^{n} a_i^2) \qquad \text{由规则 b)}$$

这样,我们就推出了重要的恒等式

$$\sum_{i=0}^{n}\sum_{j=0}^{i} a_i a_j = \frac{1}{2}((\sum_{i=0}^{n} a_i)^2 + (\sum_{i=0}^{n} a_i^2)) \tag{13}$$

例 3(几何级数的和)　假定 $x \neq 1, n \geqslant 0$,则

$$a + ax + \cdots + ax^n = \sum_{0 \leqslant j \leqslant n} ax^j = \qquad \text{由定义(2)}$$

$$a + \sum_{1 \leqslant j \leqslant n} ax^j = \qquad \text{由规则 d)}$$

$$a + x\sum_{1 \leqslant j \leqslant n} ax^{j-1} = \qquad \text{由 a) 的一个非常特殊的情况}$$

$$a + x\sum_{0 \leqslant j \leqslant n-1} ax^j = \qquad \text{由规则 b)[参看等式(6)]}$$

$$a + x\sum_{0 \leqslant j \leqslant n} ax^j - ax^{n+1} \qquad \text{由规则 d)}$$

把第一个关系式同最后一个关系式作比较,我们有

$$(1 - x)\sum_{0 \leqslant j \leqslant n} ax^j = a - ax^{n+1}$$

因此我们得到基本公式

$$\sum_{0 \leqslant j \leqslant n} ax^j = a\left(\frac{1 - x^{n+1}}{1 - x}\right) \tag{14}$$

例 4(算术级数的和)　假定 $n \geqslant 0$,则

$$a + (a + b) + \cdots + (a + nb) =$$

$$\sum_{0 \leqslant j \leqslant n} (a + bj) = \qquad \text{由定义(2)}$$

$$\sum_{0 \leqslant n-j \leqslant n} (a + b(n - j)) = \qquad \text{由规则 b)}$$

$$\sum_{0 \leqslant j \leqslant n} (a + bn - bj) = \qquad \text{通过简化}$$

$$\sum_{0 \leqslant j \leqslant n} (2a + bn) - \sum_{0 \leqslant j \leqslant n} (a + bj) = \qquad \text{由等式(8)}$$

$$(n + 1)(2a + bn) - \sum_{0 \leqslant j \leqslant n} (a + bj)$$

这是由于第一个求和只不过是把不依赖于 j 的 $(n+1)$ 个项加在一起。现在由第一个和最后一个表达式等置而后除以 2,我们得到

$$\sum_{0\leq j\leq n}(a+bj)=a(n+1)+\frac{1}{2}bn(n+1) \tag{15}$$

这就是 $n+1$ 乘以$\frac{1}{2}(a+(a+bn))$,这可以理解为项数乘以头一项与最后一项的平均值。

注意,我们通过纯粹地使用简单的求和操作,推导出重要的等式(13),(14)和(15)。大多数的教科书可能会简单地陈述这些公式,并用归纳法来证明它们。归纳法当然是完全正确的方法;但是除了进行一些幸运的猜测之外,它首先并未启发我们如何悟出这些公式来。在对算法的分析中,我们面对着数百个求和,它们同任何明显的模式都不一致;通过像上边所做的对这些求和进行处理,我们无须进行机敏的猜测,通常就能获得答案。

如果我们采用下列括号记号,许多求和与其它公式的处理会变得简单得多。

$$[命题]=\begin{cases}1 & 如果命题为真\\0 & 如果命题为假\end{cases} \tag{16}$$

于是我们可以写出

$$\sum_{R(j)}a_j=\sum_j a_j[R(j)] \tag{17}$$

其中右边的求和是对所有整数 j 进行的,因为当 $R(j)$为假时,该无穷求和的项为零。(我们假定 a_j 对所有的j 是有定义的。)

通过括号记号我们可以用有趣的方式从规则 a)和 c)导出规则 b):

$$\begin{aligned}\sum_{R(p(j))}a_{p(j)}&=\sum_j a_{p(j)}[R(p(j))]=\\&\sum_j\sum_i a_i[R(i)][i=p(j)]=\\&\sum_i a_i[R(i)]\sum_j[i=p(j)]\end{aligned} \tag{18}$$

当 $R(i)$为真时对 j 剩下的求和等于 1,如果我们假定 p 像在(5)中所要求的那样是相关值的一个排列的话。因此留给我们的是$\sum_i a_i[R(i)]$,它是$\sum_{R(i)}a_i$。这就证明了(5)。如果 p 不是这样一个排列,(18)给出$\sum_{R(p(j))}a_{p(j)}$的真正值。

括号记号最著名的特殊情况是所谓的克罗内克 δ 符号:

$$\delta_{ij}=[i=j]=\begin{cases}1,如果\ i=j\\0,如果\ i\neq j\end{cases} \tag{19}$$

它是由 Leopold Kronecker(克罗内克)于 1868 年给出的。像(16)那样更一般的记号是由 K.E. Iverson 于 1962 年给出的;因此,(16)经常被称为 Iverson 约定[参看 D.E. Knuth, *AMM* **99** (1992),403 ~ 422]。

类似于求和符号,有一个乘积符号

$$\prod_{R(j)} a_j \tag{20}$$

表示所有 a_j 的乘积，对于这些 a_j，整数 j 满足 $R(j)$。如果没有这样的整数 j 存在，就定义此乘积的值为 1(不是 0)。

只须通过适当的简单修改，运算 b)，c)和 d)如同对于"$\sum$"一样，对于"$\prod$"也正确。本节末尾的习题给出了使用乘积符号的一些例子。

在结束本小节之前，我们提出用于多重求和的另一个方便的记法：一个$\sum$符号可以用于若干个下标变量中的一个或多个关系，表示求和是对于满足条件的所有变量的组合进行。例如，

$$\sum_{0\le i\le n}\ \sum_{0\le j\le n} a_{ij} = \sum_{0\le i,j\le n} a_{ij};\quad \sum_{0\le i\le n}\ \sum_{0\le j\le i} a_{ij} = \sum_{0\le j\le i\le n} a_{ij}$$

这一记法并未表示求和中的一个下标比任何其它下标优先，因此它允许我们以一个新的方式来推导(10)式：

$$\sum_{i=1}^{n}\sum_{j=1}^{i} a_{ij} = \sum_{i,j} a_{ij}[1\le i\le n][1\le j\le i] =$$
$$\sum_{i,j} a_{ij}[1\le j\le n][j\le i\le n] = \sum_{j=1}^{n}\sum_{i=j}^{n} a_{ij}$$

这里使用了$[1\le i\le n][1\le j\le i]=[1\le j\le i\le n]=[1\le j\le n][j\le i\le n]$这个事实。根据等式

$$[R(i)][S(i,j)] = [R(i) \text{ 且 } S(i,j)] = [S'(j)][R'(i,j)] \tag{21}$$

也可以类似地导出更一般的等式(9)。

为了说明带有若干个下标的求和的有用性，我们举一个进一步的例子：

$$\sum_{\substack{j_1+\cdots+j_n=n\\ j_1\ge\cdots\ge j_n\ge 0}} a_{j_1\cdots j_n} \tag{22}$$

其中 a 是以 n 元组为下标的变量。例如，如果 $n=5$，则它表示

$$a_{11111}+a_{21110}+a_{22100}+a_{31100}+a_{32000}+a_{41000}+a_{50000}$$

(请参看 1.2.1 小节关于一个数的分划的讨论。)

习 题——第一组

1. [*01*] 如果 $n=3.14$，记号$\sum_{1\le j\le n} a_j$ 的意思是什么？
2. [*10*] 不使用$\sum$符号，分别写出与

$$\sum_{0\le n\le 5}\frac{1}{2n+1}$$

和

$$\sum_{0\le n^2\le 5}\frac{1}{2n^2+1}$$

相等价的式子。

▶ 3. [*13*] 说明为什么尽管有规则 b)，但上题的两个结果却是不同的。

4. [*10*] 不使用$\sum$符号，对于 $n=3$ 的情况把等式(10)两边的等价式写成一些和数之和。

▶ 5. [*HM20*] 假定级数收敛，证明规则 a)对于任何无穷级数都成立。

6. [*HM20*] 假定四个和数中的任意三个存在，证明规则 d)对于一个任意的无穷级数成立。

7. [*HM23*] 假定 c 是一个整数，证明即使两个级数都是无穷级数，$\sum_{R(j)}a_j=\sum_{R(c-j)}a_{c-j}$也成立。

8. [*HM25*] 举出一个使等式(7)为假的无穷级数的例子。

▶ 9. [*05*] 如果 $n=-1$，等式(14)的推导是否成立？

10. [*05*] 如果 $n=-2$，等式(14)的推导是否成立？

11. [*03*] 如果 $x=1$，等式(14)的右边应当是什么？

12. [*10*] $1+\frac{1}{7}+\frac{1}{49}+\frac{1}{343}+\cdots+(\frac{1}{7})^n$ 等于什么？

13. [*10*] 利用等式(15)，并假定 $m\le n$，计算$\sum_{j=m}^{n}j$。

14. [*11*] 利用上题的结果，计算$\sum_{j=m}^{n}\sum_{k=r}^{s}jk$。

▶ 15. [*M22*] 对于小的 n 值计算和数 $1\times2+2\times2^2+3\times2^3+\cdots+n\times2^n$。你是否已看出这些数所展示的形式？如未看出，通过类似于导出等式(14)的处理来发现它。

16. [*M22*] 如果 $x\ne1$，不用数学归纳法，证明

$$\sum_{j=0}^{n}jx^j=\frac{nx^{n+2}-(n+1)x^{n+1}+x}{(x-1)^2}$$

▶ 17. [*M00*] 设 S 是整数的一个集合，问$\sum_{j\in S}1$ 等于什么？

18. [*M20*] 假定 $R(i)$是关系"n 是 i 的倍数"，$S(i,j)$是关系"$1\le j<i$"，说明如何像等式(9)一样交换求和次序。

▶ 19. [*20*] $\sum_{j=m}^{n}(a_j-a_{j-1})$等于什么？

▶ 20. [*25*] I.J.Matrix 博士发现了一个值得注意的公式序列：

$9\times1+2=11, 9\times12+3=111, 9\times123+4=1111, 9\times1234+5=11111$

a)借助于$\sum$符号写出博士的伟大发现。

b)你对 a)的回答无疑是以数 10 为底的十进制系统进行的；推广这一公式使得你将得出或许对于任何底 b 都有效的公式。

c)使用正文中导出的公式或者上边的(16)题，证明在 b)中你导出的公式。

▶ 21. [*M25*] 利用(8)和(17)推导出规则 d)。

▶ 22. [*20*] 关于式(5)，(7)，(8)和(11)，试指出相似的积。

23. [*10*] 说明为什么当没有整数满足 $R(j)$时，分别定义$\sum_{R(j)}a_j$ 和$\prod_{R(j)}a_j$ 为 0 和为 1 是好的想法？

24. [*20*] 假设 $R(j)$仅对有限多个 j 为真。假定所有 $a_j>0$,试对满足 $R(j)$的整数的个数使用归纳法,证明 $\log_b\prod_{R(j)}a_j=\sum_{R(j)}(\log_b a_j)$。

▶ 25. [*15*] 考虑下面的推导有无差错:

$$\Big(\sum_{i=1}^{n}a_i\Big)\Big(\sum_{j=1}^{n}\frac{1}{a_j}\Big)=\sum_{1\le i\le n}\ \sum_{1\le j\le n}\frac{a_i}{a_j}=\sum_{1\le i\le n}\ \sum_{1\le i\le n}\frac{a_i}{a_i}=\sum_{i=1}^{n}1=n$$

26. [*25*] 通过如习题 22 中所述的对"$\prod$"的处理,证明$\prod_{i=0}^{n}\prod_{j=0}^{i}a_ia_j$ 可借助于$\prod_{i=0}^{n}a_i$ 来表达。

27. [*M20*] 假定 $0<a_j<1$,通过证明

$$\prod_{j=1}^{n}(1-a_j)\ge 1-\sum_{j=1}^{n}a_j$$

推广习题 1.2.1-9 中的结果。

28. [*M22*] 求出$\prod_{j=2}^{n}(1-1/j^2)$的一个简单公式。

▶ 29. [*M30*] (a) 借助于本小节末尾所说明的多重求和记号,表达$\sum_{i=0}^{n}\sum_{j=0}^{i}\sum_{k=0}^{j}a_ia_ja_k$。(b) 借助于$\sum_{i=0}^{n}a_i$,$\sum_{i=0}^{n}a_i^2$ 和$\sum_{i=0}^{n}a_i^3$,表达同一个和数[参看等式(13)]。

▶ 30. [*M23*] (J. Binet(比内),1812) 不用归纳法,证明等式

$$\Big(\sum_{j=1}^{n}a_jx_j\Big)\Big(\sum_{j=1}^{n}b_jy_j\Big)=\Big(\sum_{j=1}^{n}a_jy_j\Big)\Big(\sum_{j=1}^{n}b_jx_j\Big)+\sum_{1\le j<k\le n}(a_jb_k-a_kb_j)(x_jy_k-x_ky_j)$$

[当 $w_1,\cdots,w_n,z_1,\cdots,z_n$ 为任意复数时出现一个重要的特殊情况,我们置 $a_j=w_j$, $b_j=\bar z_j$, $x_j=\bar w_j$, $y_j=z_j$:

$$\Big(\sum_{j=1}^{n}|w_j|^2\Big)\Big(\sum_{j=1}^{n}|z_j|^2\Big)=\Big|\sum_{j=1}^{n}w_jz_j\Big|^2+\sum_{1\le j<k\le n}|w_j\bar z_k-w_k\bar z_j|^2$$

项$|w_j\bar z_k-w_k\bar z_j|^2$是非负的,因此著名的柯西—施瓦茨(Cauchy-Schwarz)不等式

$$\Big(\sum_{j=1}^{n}|w_j|^2\Big)\Big(\sum_{j=1}^{n}|z_j|^2\Big)\ge\Big|\sum_{j=1}^{n}w_jz_j\Big|^2$$

是比内公式的一个推论。]

31. [*M20*] 借助于$\sum_{j=1}^{n}u_jv_j$,$\sum_{j=1}^{n}u_j$ 和$\sum_{j=1}^{n}v_j$,利用比内公式表达和数

$$\sum\nolimits_{1\le j<k\le n}(u_j-u_k)(v_j-v_k)$$

32. [*M20*] 证明

$$\prod_{j=1}^{n}\sum_{i=1}^{m}a_{ij}=\sum_{1\le i_1,\cdots,i_n\le m}a_{i_1 1}\cdots a_{i_n n}$$

▶ 33. [*M30*] 一个傍晚,Matrix 博士发现了一些公式,这些公式甚至可以归入比习题 20 中的公式更为值得注意的类型:

$$\frac{1}{(a-b)(a-c)}+\frac{1}{(b-a)(b-c)}+\frac{1}{(c-a)(c-b)}=0$$

$$\frac{a}{(a-b)(a-c)}+\frac{b}{(b-a)(b-c)}+\frac{c}{(c-a)(c-b)}=0$$

$$\frac{a^2}{(a-b)(a-c)}+\frac{b^2}{(b-a)(b-c)}+\frac{c^2}{(c-a)(c-b)}=1$$

$$\frac{a^3}{(a-b)(a-c)}+\frac{b^3}{(b-a)(b-c)}+\frac{c^3}{(c-a)(c-b)}=a+b+c$$

证明这些公式是一个一般定律的特殊情况;设 $x_1,x_2,\cdots,x_n$ 是不同的数,证明

$$\sum_{j=1}^{n}\Big(x_j^r\Big/\prod_{\substack{1\le k\le n\\ k\ne j}}(x_j-x_k)\Big)=\begin{cases}0, & \text{如果 } 0\le r<n-1\\ 1, & \text{如果 } r=n-1\\ \sum_{j=1}^{n}x_j, & \text{如果 } r=n\end{cases}$$

34. [*M25*] 假定 $1\le m\le n$ 且 x 任意,证明

$$\sum_{k=1}^{n}\frac{\prod_{1\le r\le n,\,r\ne m}(x+k-r)}{\prod_{1\le r\le n,\,r\ne k}(k-r)}=1$$

例如,如果 $n=4$ 和 $m=2$,则

$$\frac{x(x-2)(x-3)}{(-1)(-2)(-3)}+\frac{(x+1)(x-1)(x-2)}{(1)(-1)(-2)}+$$

$$\frac{(x+2)x(x-1)}{(2)(1)(-1)}+\frac{(x+3)(x+1)x}{(3)(2)(1)}=1$$

35. [*HM20*] 记号 $\sup_{R(j)}a_j$ 以完全类似于 Σ 和 Π 符号的方式,用来表示元素 a_j 的最小上界。(当 $R(j)$ 仅为有限多个 j 所满足时,通常使用记号 $\max_{R(j)}a_j$ 来表示同一个量。)说明如何可使规则 a),b),c)和 d)适合于对这个记号的处理。特别是,讨论以下的类似于规则 a)的规则:

$$(\sup_{R(i)}a_i)+(\sup_{S(j)}b_j)=\sup_{R(i)}(\sup_{S(j)}(a_i+b_j))$$

而且当没有 j 满足 $R(j)$ 时,给出这个记号的适当定义。

习　题——第二组

行列式和矩阵　下列有趣的问题是提供给那些至少对行列式和初等矩阵理论已经入门并有些经验的读者的。一个行列式可以通过灵活地组合以下几种运算来计算:(a)从一行或一列分解出一个量;(b)把一行(或一列)的倍数加到另一行(或一列)上;(c)按余因子(cofactor)展开。运算(c)最简单和最常用的形式是,假定左上角的元素是 +1 而在整个头一行或者整个头一列中剩下的元素全为 0,则简单地删去整个头一行和整个头一列;然后计算所得到的较小的行列式。一般说来,在一个 $n\times n$ 行列式中一个元素 a_{ij} 的余因子是 $(-1)^{i+j}$ 乘以删去 a_{ij} 在其中出现的行和列所

得到的$(n-1)\times(n-1)$行列式。一个行列式的值等于$\sum a_{ij}\mathrm{cofactor}(a_{ij})$,其中求和是这样进行的,即 i 或 j 之一保持不变,而另一个下标则从 1 变到 n。

如果(b_{ij})是矩阵(a_{ij})的逆,则 b_{ij}等于 a_{ji}(不是 a_{ij})的余因子除以整个原矩阵的行列式。

下列类型的矩阵有特殊的重要性:

范德蒙德(Vandermonde)矩阵

$$a_{ij} = x_j^i$$

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & & \vdots \\ x_1^n & x_2^n & \cdots & x_n^n \end{pmatrix}$$

组合矩阵

$$a_{ij} = y + \delta_{ij} x$$

$$\begin{pmatrix} x+y & y & \cdots & y \\ y & x+y & \cdots & y \\ \vdots & \vdots & & \vdots \\ y & y & \cdots & x+y \end{pmatrix}$$

柯西矩阵

$$a_{ij} = 1/(x_i + y_j)$$

$$\begin{pmatrix} 1/(x_1+y_1) & 1/(x_1+y_2) & \cdots & 1/(x_1+y_n) \\ 1/(x_2+y_1) & 1/(x_2+y_2) & \cdots & 1/(x_2+y_n) \\ \vdots & \vdots & & \vdots \\ 1/(x_n+y_1) & 1/(x_n+y_2) & \cdots & 1/(x_n+y_n) \end{pmatrix}$$

36. [*M23*] 证明组合矩阵的行列式是 $x^{n-1}(x+ny)$。

▶ 37. [*M24*] 证明范德蒙德矩阵的行列式是

$$\prod_{1\le j\le n} x_j \prod_{1\le i<j\le n} (x_j - x_i)$$

▶ 38. [*M25*] 证明柯西矩阵的行列式是

$$\prod_{1\le i<j\le n} (x_j - x_i)(y_j - y_i) \Big/ \prod_{1\le i,j\le n} (x_i + y_j)$$

39. [*M23*] 证明组合矩阵的逆矩阵是以元素 $b_{ij}=(-y+\delta_{ij}(x+ny))/x(x+ny)$组成的组合矩阵。

40. [*M20*] 证明范德蒙德矩阵的逆是由

$$b_{ij} = \Big(\sum_{\substack{1\le k_1<\cdots<k_{n-j}\le n \\ k_1,\ldots,k_{n-j}\ne i}} (-1)^{j-1} x_{k_1}\cdots x_{k_{n-j}} \Big) \Big/ x_i \prod_{\substack{1\le k\le n \\ k\ne i}} (x_k - x_i)$$

给出的,不要为分子中复杂的求和所吓倒——它只不过是在多项式$(x_1-x)\cdots(x_n-x)/(x_i-x)$中 x^{j-1}的系数。

41. [*M26*] 证明柯西矩阵的逆由

$$b_{ij} = \Big(\prod_{1\le k\le n} (x_j + y_k)(x_k + y_i) \Big) \Big/ (x_j + y_i) \Big(\prod_{\substack{1\le k\le n \\ k\ne j}} (x_j - x_k) \Big) \Big(\prod_{\substack{1\le k\le n \\ k\ne i}} (y_i - y_k) \Big)$$

给出。

42. [*M18*] 在组合矩阵的逆中全部 n^2 个元素之和是什么?

43. [*M24*] 在范德蒙德矩阵的逆中全部 n^2 个元素之和是什么?

▶ 44. [*M26*] 在柯西矩阵的逆中全部 n^2 个元素之和是什么?

▶ 45. [*M25*] 希尔伯特(Hilbert)矩阵,有时称做(无穷的)希尔伯特矩阵的 $n\times n$ 段,是一个元素为 $a_{ij}=1/(i+j-1)$的矩阵。证明这是柯西矩阵的一个特殊情况,试求其逆,证明这个逆的每个元素是一个整数,而且证明逆的全部元素之和是 n^2。(注:希尔伯特矩阵经常用来检验各种矩阵操作的算法。因为它们在数值上是不稳定的,而且它们有已知的逆。然而,试图比较在本习题中给出的已知的逆同一个希尔伯特矩阵计算出来的逆是不对的,因为要加以求逆的矩阵事先必须以已舍入的数表示。由于出现不稳定性,一个近似的希尔伯特矩阵的逆将稍稍不同于精确的希尔伯特矩阵之逆。由于逆矩阵的元素是整数,而且由于逆矩阵和原来的矩阵一样不稳定,这个逆矩阵可以精确地加以表述,因而人们可以试图对这个逆来求逆。然而,在逆中出现的整数十分大。)对此问题的解需要有关于阶乘和二项式系数的初步知识,在 1.2.5 和 1.2.6 小节中要讨论这些知识。

▶ 46. [*M30*] 设 A 是一个 $m\times n$ 矩阵,B 是一个 $n\times m$ 矩阵,假定 $1\leqslant j_1,j_2,\cdots,j_m\leqslant n$,命 $A_{j_1j_2\cdots j_m}$表示由 A 的 $j_1,j_2,\cdots,j_m$ 列组成的 $m\times m$ 矩阵,命 $B_{j_1j_2\cdots j_m}$表示由 B 的 $j_1,\cdots,j_m$ 行组成的 $m\times m$ 矩阵,证明比内—柯西恒等式

$$\det(AB)=\sum_{1\leqslant j_1<j_2<\cdots<j_m\leqslant n}\det(A_{j_1j_2\cdots j_m})\det(B_{j_1j_2\cdots j_m})$$

(注意特殊情况:(i) $m=n$,(ii) $m=1$,(iii) $B=A^{\mathrm T}$,(iv) $m>n$,(v) $m=2$。)

47. [*M27*](C. Krattenthaler) 证明

$$\det\begin{pmatrix}(x+q_2)(x+q_3) & (x+p_1)(x+q_3) & (x+p_1)(x+p_2)\\ (y+q_2)(y+q_3) & (y+p_1)(y+q_3) & (y+p_1)(y+p_2)\\ (z+q_2)(z+q_3) & (z+p_1)(z+q_3) & (z+p_1)(z+p_2)\end{pmatrix}=$$

$$(x-y)(x-z)(y-z)(p_1-q_2)(p_1-q_3)(p_2-q_3)$$

并把这一等式推广成有 $3n-2$ 个变量 $x_1,\cdots,x_n,p_1,\cdots,p_{n-1},q_2,\cdots,q_n$ 的 $n\times n$ 行列式的一个恒等式。把你的公式同习题 38 的结果加以比较。

1.2.4 整数函数和初等数论

如果 x 是任意实数,则写

$\lfloor x\rfloor$ = 小于或等于 x 的最大整数(x 的底限);

$\lceil x\rceil$ = 大于或等于 x 的最小整数(x 的顶限)。

1970 年以前经常使用记号[x]来表示这些函数之一,通常是表示前一个。但是由 K. E. Ivevson 于 20 世纪 60 年代引进的上边的记号更为有用,因为$\lfloor x\rfloor$和$\lceil x\rceil$实际上几乎同样经常地出现。函数$\lfloor x\rfloor$有时叫做 entier 函数,此词取自于法文的“整数”一词。

容易验证下列的公式和例子:

$$\lfloor\sqrt{2}\rfloor = 1, \lceil\sqrt{2}\rceil = 2, \lfloor +\frac{1}{2}\rfloor = 0, \lceil -\frac{1}{2}\rceil = 0, \lfloor -\frac{1}{2}\rfloor = -1 (非 0!)$$

$$\lceil x\rceil = \lfloor x\rfloor \qquad 当且仅当 x 为整数时$$

$$\lceil x\rceil = \lfloor x\rfloor + 1 \qquad 当且仅当 x 不是整数时$$

$$\lfloor -x\rfloor = -\lceil x\rceil$$

$$x - 1 < \lfloor x\rfloor \leqslant x \leqslant \lceil x\rceil < x + 1$$

本小节末尾的习题列出了关于底限和顶限运算的其它重要公式。

如果 x 和 y 是任意实数,则我们定义以下的二元运算。

$$x \bmod y = x - y\lfloor x/y\rfloor, \qquad 如果\ y \neq 0; \qquad x \bmod 0 = x \tag{1}$$

从这个定义,我们看出,当 $y \neq 0$ 时,

$$0 \leqslant \frac{x}{y} - \left\lfloor \frac{x}{y} \right\rfloor = \frac{x \bmod y}{y} < 1 \tag{2}$$

因此,

a)如果 $y > 0$,则 $0 \leqslant x \bmod y < y$;

b)如果 $y < 0$,则 $0 \geqslant x \bmod y > y$;

c)量 $x - (x \bmod y)$是 y 的整数倍。

我们称 $x \bmod y$ 为 y 除 x 的余数;类似地,我们称$\lfloor x/y\rfloor$为商。

当 x 和 y 是整数时,"mod"因此是一个熟知的运算:

$$5 \bmod 3 = 2 \qquad 18 \bmod 3 = 0 \qquad -2 \bmod 3 = 1 \tag{3}$$

当且仅当 x 是 y 的倍数,即当且仅当 x 可为 y 所整除时,我们有 $x \bmod y = 0$。记号 $y \backslash x$,读作"y 整除 x",指的是 y 是正整数,而且 $x \bmod y = 0$。

当 x 和 y 取任意实数值时,"mod"运算也是有用的。例如,对于三角函数,我们可以写

$$\tan x = \tan(x \bmod \pi)$$

量 $x \bmod 1$ 是 x 的小数部分;由等式(1),

$$x = \lfloor x\rfloor + (x \bmod 1) \tag{4}$$

数论的作者们经常以不同的但是密切相关的意义来使用"mod"这一缩写。我们将使用以下的形式来表达数论中的同余概念:命题

$$x \equiv y (\text{modulo } z) \tag{5}$$

指的是 $x \bmod z = y \bmod z$;这等于是说 $x - y$ 是 z 的整数倍。表达式(5)读作"x 与 y 模 z 同余"。

现在让我们来指出同余的基本的初等性质，这些性质将在本书的数论论述中用到。假设下列公式中的所有变量都是整数。如果两个整数 x 和 y 没有公因子，即如果它们的最大公因子为 1，则说它们是*互素*的。在这样的情况下，我们写 $x \perp y$。互素整数的概念是一个熟知的概念，因为当分子和分母互素时，习惯上就说一个分数是“最简的”。

定律 A 如果 $a \equiv b$ 和 $x \equiv y$，则 $a \pm x \equiv b \pm y$，且 $ax \equiv by \pmod{m}$。

定律 B 如果 $ax \equiv by$ 和 $a \equiv b$，且如果 $a \perp m$，则 $x \equiv y \pmod{m}$。

定律 C $a \equiv b \pmod{m}$ 当且仅当 $an \equiv bn \pmod{mn}$，当 $n \neq 0$ 时。

定律 D 如果 $r \perp s$，则 $a \equiv b \pmod{rs}$，当且仅当 $a \equiv b \pmod{r}$ 和 $a \equiv b \pmod{s}$。

定律 A 指出，我们可以在模 m 下做加法、减法和乘法，就如同我们做通常的加法、减法和乘法那样。定律 B 考虑除法运算并说明，当因子同模数互素时，我们也可以除掉公因子。定律 C 和 D 考虑当改变模数时，发生什么情况。这些定律将在下边的习题中被证明。

下列重要定理是定律 A 和定律 B 的一个推论。

定理 F（费马定理，1640） *如果 p 是素数，则对于所有整数 a，$a^p \equiv a \pmod{p}$。*

证明 如果 a 是 p 的倍数，显然 $a^p \equiv 0 \equiv a \pmod{p}$。所以我们只须考虑 $a \bmod p \neq 0$ 的情况。因为 p 是一个素数，这意味着 $a \perp p$。考虑数

$$0 \bmod p,\ a \bmod p,\ 2a \bmod p,\ \cdots,\ (p-1)a \bmod p \tag{6}$$

这 p 个数都是不同的，因为如果 $ax \bmod p = ay \bmod p$，则由定义(5)，$ax \equiv ay \pmod{p}$；因此由定律 B，$x \equiv y \pmod{p}$。

由于(6)给出 p 个不同的数，所有这些数还都非负和小于 p，我们看出，头一个数是零，而剩下的是在某个顺序下的整数 $1, 2, \cdots, p-1$。因此由定律 A，

$$(a)(2a)\cdots((p-1)a) \equiv 1 \cdot 2 \cdots (p-1) \pmod{p} \tag{7}$$

用 a 来乘这个同余式的每边我们得到

$$a^p(1 \cdot 2 \cdots (p-1)) \equiv a(1 \cdot 2 \cdots (p-1)) \pmod{p} \tag{8}$$

这就证明了定理，因为因式 $1, 2, \cdots, p-1$ 的每一个都同 p 互素，故可由定律 B 将其删去。▌

习 题

1. [*00*] 什么是 $\lfloor 1.1 \rfloor$，$\lfloor -1.1 \rfloor$，$\lceil -1.1 \rceil$，$\lfloor 0.99999 \rfloor$ 和 $\lfloor \lg 35 \rfloor$？

▶ 2. [*01*] 什么是 $\lceil \lfloor x \rfloor \rceil$？

3. [*M10*] 设 n 是一个整数，且设 x 是一个实数。证明

a) $\lfloor x \rfloor < n$ 当且仅当 $x < n$； b) $n \leq \lfloor x \rfloor$ 当且仅当 $n \leq x$；

c) $\lceil x \rceil \leq n$ 当且仅当 $x \leq n$； d) $n < \lceil x \rceil$ 当且仅当 $n < x$；

e) $\lfloor x \rfloor = n$ 当且仅当 $x-1<n\leqslant x$，又当且仅当 $n\leqslant x<n+1$；

f) $\lceil x \rceil = n$ 当且仅当 $x\leqslant n<x+1$，又当且仅当 $n-1<x\leqslant n$。

(这些公式是证明关于 $\lfloor x \rfloor$ 和 $\lceil x \rceil$ 的事实的最重要工具。)

▶ 4. [*M10*] 利用前边的习题，证明 $\lfloor -x \rfloor = -\lceil x \rceil$

5. [*16*] 假定 x 是正实数，指出表达把 x 舍入成为最接近的整数的一个简单公式。所希望的舍入规则是当 $x \bmod 1<\frac{1}{2}$ 时产生 $\lfloor x \rfloor$，而当 $x \bmod 1\geqslant\frac{1}{2}$ 时产生 $\lceil x \rceil$。你的答案应当是包括这两种情况的一个公式。试讨论当 x 是负数时由你的公式所得到的舍入。

▶ 6. [*20*] 对于所有的实数 x，下列哪些等式为真？

(a) $\lfloor\sqrt{\lfloor x \rfloor}\rfloor=\lfloor\sqrt{x}\rfloor$；(b) $\lceil\sqrt{\lceil x \rceil}\rceil=\lceil\sqrt{x}\rceil$；(c) $\lceil\sqrt{\lfloor x \rfloor}\rceil=\lceil\sqrt{x}\rceil$

7. [*M15*] 证明 $\lfloor x \rfloor+\lfloor y \rfloor\leqslant\lfloor x+y \rfloor$ 而且当且仅当 $x \bmod 1+y \bmod 1<1$ 时等式成立。对于顶限有类似的公式成立吗？

8. [*00*] $100 \bmod 3$，$100 \bmod 7$，$-100 \bmod 7$，$-100 \bmod 0$ 等于什么？

9. [*05*] $5 \bmod -3$，$18 \bmod -3$，$-2 \bmod -3$ 等于什么？

▶ 10. [*10*] $1.1 \bmod 1$，$0.11 \bmod .1$，$0.11 \bmod -.1$ 等于什么？

11. [*00*] 按照我们的约定“$x\equiv y \pmod{0}$”意味着什么？

12. [*00*] 什么整数与 1 互素？

13. [*M00*] 根据约定，我们说 0 和 n 的最大公因子是 $|n|$，什么整数与 0 互素？

▶ 14. [*12*] 如果 $x \bmod 3=2$ 和 $x \bmod 5=3$，则 $x \bmod 15$ 等于什么？

15. [*10*] 证明 $z(x \bmod y)=(zx)\bmod(zy)$。[定律 C 是这个分配律的一个直接推论。]

16. [*M10*] 假设 $y>0$。证明如果 $(x-z)/y$ 是一个整数且如果 $0\leqslant z<y$，则 $z=x \bmod y$。

17. [*M15*] 直接从同余的定义证明定律 A，并且证明定律 D 的一半：如果 $a\equiv b \pmod{rs}$，则 $a\equiv b \pmod{r}$ 和 $a\equiv b \pmod{s}$。(这里 r 和 s 是任意整数。)

18. [*M15*] 用定律 B，证明定律 D 的另一半：假设 $r\perp s$，如果 $a\equiv b \pmod{r}$ 和 $a\equiv b \pmod{s}$，则 $a\equiv b \pmod{rs}$。

▶ 19. [*M10*]（关于逆的定律）如果 $n\perp m$，则存在一个整数 n' 使得 $nn'\equiv 1 \pmod{m}$。利用扩充的欧几里得算法(算法 1.2.1E)来证明这一点。

20. [*M15*] 用关于逆的定律和定律 A 来证明定律 B。

21. [*M22*]（算术基本定理）用定律 B 和习题 1.2.1-5 证明，每个整数 $n>1$ 可以惟一地表示成素数的乘积(除了其因子的次序可以颠倒外)。换言之，恰有一个方式把 n 写成 $n=p_1p_2\cdots p_k$，其中各 p_j 是素数，且 $p_1\leqslant p_2\leqslant\cdots\leqslant p_k$。

▶ 22. [*M10*] 举出一个例子说明，如果 a 与 m 不互素，定律 B 不总成立。

23. [*M10*] 举出一个例子说明，如果 r 与 s 不互素，定律 D 不总成立。

▶ 24. [*M20*] 定律 A，B，C 和 D 推广到什么程度，才能适用于任何实数而不仅是整数。

25. [*M02*] 根据定理 F，证明，当 p 是素数时，$a^{p-1} \bmod p=[a$ 不是 p 的倍数$]$。

26. [*M15*] 设 p 是奇素数，设 a 是任意整数，且设 $b=a^{(p-1)/2}$。证明 $b \bmod p$ 或等于 0，或等于 1，或等于 $p-1$。[提示：考虑 $(b+1)(b-1)$。]

27. [*M15*] 假定 n 是一个正整数，设 $\varphi(n)$ 是 $\{0,1,\cdots,n-1\}$ 当中与 n 互素的数的个数。于是 $\varphi(1)=1$，$\varphi(2)=1$，$\varphi(3)=2$，$\varphi(4)=2$，等等。证明，如果 p 是素数，则 $\varphi(p)=p-1$；并请计算当 e 是正整数时，$\varphi(p^e)$ 的值。

▶ 28. [*M25*] 证明，证明定理 F 的方法也可以用来证明以下的(定理 F 的)推广，这个推广叫做

欧拉定理：对于任意整数 m，当 $a\perp m$ 时，$a^{\varphi(m)}\equiv 1\pmod{m}$。（特别是，习题 19 中的数 n' 可以取作 $n^{\varphi(m)-1} \bmod m$。）

29. [*M22*] 正整数 n 的一个函数 $f(n)$ 叫做乘性的，如果当 $r\perp s$ 时，$f(rs)=f(r)\cdot f(s)$。证明以下各函数是乘性的：(a) $f(n)=n^c$，其中 c 是任意常数；(b) $f(n)=[$对于任意整数 $k>1$，n 不可为 k^2 所整除$]$；(c) $f(n)=c^k$，其中 k 是整除 n 的不同素数的个数；(d) 任意两个乘性函数的乘积。

30. [*M30*] 证明习题 27 的函数 $\varphi(n)$ 是乘性的，利用这一事实，计算 $\varphi(1000000)$。并且对于 n 已经分解成素数的情形，给出以简单方式计算 $\varphi(n)$ 的一个方法。

31. [*M22*] 证明，如果 $f(n)$ 是乘性的，则 $g(n)=\sum_{d\backslash n}f(d)$ 也是乘性的。

32. [*M18*] 对于任意函数 $f(x,y)$，证明双重求和恒等式

$$\sum_{d\backslash n}\sum_{c\backslash d}f(c,d)=\sum_{c\backslash n}\sum_{d\backslash(n/c)}f(c,cd)$$

33. [*M18*] 假定 m 和 n 是整数，试计算 (a) $\lfloor\frac{1}{2}(n+m)\rfloor+\lfloor\frac{1}{2}(n-m+1)\rfloor$；(b) $\lceil\frac{1}{2}(n+m)\rceil+\lceil\frac{1}{2}(n-m+1)\rceil$。（$m=0$ 的特殊情况值得注意。）

▶ 34. [*M21*] 为保证对于所有实数 $x\geq 1$，$\lfloor\log_b x\rfloor=\lfloor\log_b\lfloor x\rfloor\rfloor$ 都成立，对实数 $b>1$ 应当加上什么条件才是必要和充分的？

▶ 35. [*M20*] 假定 m 和 n 是整数且 $n>0$，证明对所有实数 x，

$$\lfloor(x+m)/n\rfloor=\lfloor(\lfloor x\rfloor+m)/n\rfloor$$

（当 $m=0$ 时，我们有一个重要的特殊情况。）请问对于顶限函数，类似的结果是否成立？

36. [*M23*] 证明 $\sum_{k=1}^{n}\lfloor k/2\rfloor=\lfloor n^2/4\rfloor$；并计算 $\sum_{k=1}^{n}\lceil k/2\rceil$。

▶ 37. [*M30*] 设 m 和 n 是整数，$n>0$。证明

$$\sum_{0\leq k<n}\left\lfloor\frac{mk+x}{n}\right\rfloor=\frac{(m-1)(n-1)}{2}+\frac{d-1}{2}+d\lfloor x/d\rfloor$$

其中 d 是 m 和 n 的最大公因子，而 x 是任意实数。

38. [*M26*]（E. Busche, 1909）证明对所有实数 x 与 y，且 $y>0$，

$$\sum_{0\leq k<y}\left\lfloor x+\frac{k}{y}\right\rfloor=\lfloor xy+\lfloor x+1\rfloor(\lceil y\rceil-y)\rfloor$$

特别是当 y 是正整数 n 时，我们有重要的公式

$$\lfloor x\rfloor+\left\lfloor x+\frac{1}{n}\right\rfloor+\cdots+\left\lfloor x+\frac{n-1}{n}\right\rfloor=\lfloor nx\rfloor$$

39. [*HM35*] 一个函数 f，当 n 是正整数时，如果有

$$f(x)+f(x+\frac{1}{n})+\cdots+f(x+\frac{n-1}{n})=f(nx)$$

则叫做重叠函数，上题表明 $\lfloor x\rfloor$ 是重叠函数，证明下列函数也是重叠的：

a) $f(x)=x-\frac{1}{2}$；

b) $f(x)=[x$ 是整数$]$；

c) $f(x)=[x$ 是正整数$]$;

d) $f(x)=[$存在一个有理数 r 和一个整数 m 使得 $x=r\pi+m]$;

e) 和 d)中的函数类似的三个其它的函数,限制 r 和/或 m 为正值;

f) 如果允许 $f(x)=-\infty$,$f(x)=\log|2\sin\pi x|$;

g) 任意两个重叠函数之和;

h) 一个重叠函数的常数倍;

i) 函数 $g(x)=f(x-\lfloor x\rfloor)$,其中 $f(x)$是重叠函数。

40. [*HM46*] 研究重叠函数的类:确定一种特殊类型的所有重叠函数。例如,习题 39 的 a)中的函数是不是仅有的连续的重叠函数;研究更为一般的函数类可能是有趣的,对于它,有

$$f(x)+f(x+\frac{1}{n})+\cdots+f(x+\frac{n-1}{n})=a_nf(nx)+b_n$$

这里 a_n 和 b_n 是依赖于 n 而不依赖于 x 的数。这些函数的导数和(如果 $b_n=0$)积分也是同一类型。如果我们要求 $b_n=0$,我们有,例如,伯努利(Bernoulli)多项式,三角函数 $\cot\pi x$ 和 $\csc^2\pi x$,以及赫尔维茨(Hurwitz)广义 ζ 函数 $\zeta(s,x)=\sum_{k\geq 0}1/(k+x)^s$,对于固定的 s。对于 $b_n\neq 0$,我们还有其它熟知的函数,例如 ψ 函数。

41. [*M23*] 设 $a_1,a_2,a_3,\cdots$是序列 1,2,2,3,3,3,4,4,4,4,…;请利用底限和/或顶限函数,借助于 n,求出关于 a_n 的一个表达式。

42. [*M24*] (a) 证明

$$\sum_{k=1}^{n}a_k=na_n-\sum_{k=1}^{n-1}k(a_{k+1}-a_k),\ \text{如果}\ n>0$$

(b)上面的公式对于计算涉及底限函数的某些和数是有用的。证明,如果 b 是大于等于 2 的整数,则

$$\sum_{k=1}^{n}\lfloor\log_b k\rfloor=(n+1)\lfloor\log_b n\rfloor-(b^{\lfloor\log_b n\rfloor+1}-b)/(b-1)$$

43. [*M23*] 求$\sum_{k=1}^{n}\lfloor\sqrt{k}\rfloor$。

44. [*M24*] 证明,如果 b 和 n 是整数,$n\geq 0$,且 $b\geq 2$,$\sum_{k\geq 0}\sum_{1\leq j<b}\lfloor(n+jb^k)/b^{k+1}\rfloor=n$。当 $n<0$ 时,这个和数的值是什么?

▶ 45. [*M28*] 习题 37 的结果是多少令人惊讶的,因为它意味着

$$\sum_{0\leq k<n}\left\lfloor\frac{mk+x}{n}\right\rfloor=\sum_{0\leq k<m}\left\lfloor\frac{nk+x}{m}\right\rfloor$$

这个"互反关系"是许多类似的公式之一(参看 3.3.3 小节)。证明,对于任何函数 f,我们有

$$\sum_{0\leq j<n}f\left(\left\lfloor\frac{mj}{n}\right\rfloor\right)=\sum_{0\leq r<m}\left\lceil\frac{rn}{m}\right\rceil(f(r-1)-f(r))+nf(m-1)$$

特别是证明 $\sum_{0\leq j<n}\binom{\lfloor mj/n\rfloor+1}{k}+\sum_{0\leq j<m}\left\lceil\frac{jn}{m}\right\rceil\binom{j}{k-1}=n\binom{m}{k}$。

[提示:考虑变量 $r=\lfloor mj/n\rfloor$ 的变化。二项式系数$\binom{m}{k}$在 1.2.6 小节中讨论。]

46. [*M29*](广义互反定律) 推广习题 45 的公式以得到关于$\sum_{0\leqslant j<\alpha n}f(\lfloor mj/n\rfloor)$的一个表达式，其中 α 是任意正实数。

▶ 47. [*M31*] 当 p 是一个奇素数时，定义勒让德(Legendre)符号$\left(\frac{q}{p}\right)$为 $+1,0$ 或 -1，依赖于 $q^{(p-1)/2}\bmod p$ 是 $1,0$ 还是 $p-1$。(习题 26 证明，这些是仅有的可能的值。)

a) 假定 q 不是 p 的倍数，证明数

$$(-1)^{\lfloor 2kq/p\rfloor}(2kq \bmod p),\qquad 0<k<p/2$$

是以某种顺序与数 $2,4,\cdots,p-1$ 模 p 同余的。因此，$\left(\frac{q}{p}\right)=(-1)^{\sigma}$，其中

$$\sigma=\sum\nolimits_{0\leqslant k<p/2}\lfloor 2kq/p\rfloor$$

b) 用 a)的结果来计算$\left(\frac{2}{p}\right)$。

c) 假定 q 为奇数，证明 $\sum_{0\leqslant k<p/2}\lfloor 2kq/p\rfloor\equiv\sum_{0\leqslant k<p/2}\lfloor kq/p\rfloor$ (modulo 2)。[提示：考虑量 $\lfloor(p-1-2k)q/p\rfloor$。]

d) 假定 p 和 q 是不同的奇素数，使用习题 46 的广义互反定律以得到二次互反定律

$$\left(\frac{q}{p}\right)\left(\frac{p}{q}\right)=(-1)^{(p-1)(q-1)/4}$$

48. [*M26*] 证明或反驳下列恒等式，对于整数 m 和 n，

$$\text{(a)}\left\lfloor\frac{m+n-1}{n}\right\rfloor=\left\lceil\frac{m}{n}\right\rceil\qquad\text{(b)}\left\lfloor\frac{m+2-\lfloor n/25\rfloor}{3}\right\rfloor=\left\lfloor\frac{8n+24}{25}\right\rfloor$$

49. [*M30*] 假设整数值函数 $f(x)$满足两个简单定律：(i)$f(x+1)=f(x)+1$；(ii)对于所有正整数 n，$f(x)=f(f(nx)/n)$。证明，或者对于所有有理数 x，$f(x)=\lfloor x\rfloor$，或者对于所有有理数 x，$f(x)=\lceil x\rceil$。

1.2.5 排列和阶乘

n 个对象的排列，就是把 n 个不同的对象放在一行上的一种安排。对于三个对象 $\{a,b,c\}$有 6 种排列：

$$abc,\ acb,\ bac,\ bca,\ cab,\ cba \tag{1}$$

在算法分析中，排列的性质有着巨大的重要性，因此在本书稍后部分，我们将推演出关于它们的许多有趣的事实。* 我们的头一个任务只是计算它们：n 个对象有多少种可能的排列？有 n 种方式来选择最左边的对象，而一旦已经做出了这个选择，就还有 $n-1$ 种方式来选择一个不同的对象放置在它的下一个位置；这就为我们提供了对于头两个位置的 $n(n-1)$种选择。类似地，对于不同于头两个的第三个对象有 $n-2$ 种选

* 事实上，排列是如此重要，因此 Vaughan Pratt 建议把它们叫做 perms(原来是 permutations，——译者)，只要 Pratt 的约定被接受，计算机科学的书就会稍微薄些(或者也会稍微便宜些)。——原注

择,因此对于头三个对象总共有 $n(n-1)(n-2)$ 种可能的方式。一般说来,如果 p_{nk} 表示从 n 个对象中选择 k 个并把它们排成一行的方式数,我们看到

$$p_{nk} = n(n-1)\cdots(n-k+1) \tag{2}$$

排列的总数因此是 $p_{nn} = n(n-1)\cdots(1)$。

假设 $n-1$ 个对象的所有排列已经被构造出来,以一种归纳的方式构造 n 个对象的所有排列的过程,在我们的应用中是非常重要的。代替使用字母 $\{a,b,c\}$,我们使用数字 $\{1,2,3\}$ 来改写(1);于是排列为

$$1\,2\,3,\ 1\,3\,2,\ 2\,1\,3,\ 2\,3\,1,\ 3\,1\,2,\ 3\,2\,1 \tag{3}$$

考虑如何从这一排列得到 $\{1,2,3,4\}$ 的排列。从 $n-1$ 个对象过渡到 n 个对象,有两个主要的方法。

方法 1　对于 $\{1,2,\cdots,n-1\}$ 的每一个排列 $a_1\,a_2\ \cdots\ a_{n-1}$,通过把数 n 插入到所有可能的位置来形成 n 个排列,得到

$$n\,a_1a_2\cdots a_{n-1},\ a_1n\,a_2\cdots\ a_{n-1},\ \cdots,\ a_1a_2\cdots\ n\,a_{n-1},\ a_1\,a_2\cdots\ a_{n-1}\,n$$

例如,从(3)中的排列 2 3 1,我们得到 4 2 3 1,2 4 3 1,2 3 4 1,2 3 1 4。显然,以这种方式可得到 n 个对象的所有排列,而且没有一个排列被生成一次以上。

方法 2　对于 $\{1,2,\cdots,n-1\}$ 的每个排列 $a_1\,a_2\ \cdots\ a_{n-1}$,形成 n 个排列如下:首先构造阵列

$$a_1\,a_2\cdots\ a_{n-1}\frac{1}{2},\ a_1\,a_2\cdots\ a_{n-1}\frac{3}{2},\ \cdots,\ a_1\,a_2\cdots\ a_{n-1}(n-\frac{1}{2})$$

然后保持顺序,使用数 $\{1,2,\cdots,n\}$ 重新命名每个排列的元素。例如,从(3)中的排列 2 3 1 我们得到

$$2\,3\,1\,\frac{1}{2},\ 2\,3\,1\,\frac{3}{2},\ 2\,3\,1\,\frac{5}{2},\ 2\,3\,1\,\frac{7}{2}$$

重新命名,我们得到

$$3\,4\,2\,1,\ 3\,4\,1\,2,\ 2\,4\,1\,3,\ 2\,3\,1\,4$$

描述这一过程的另外一个方法是取排列 $a_1\,a_2\ \cdots\ a_{n-1}$ 和一个数 k,$1\leqslant k\leqslant n$;将值大于等于 k 的每一个 a_j 加 1,因此得到元素 $\{1,\cdots,k-1,k+1,\cdots,n\}$ 的一个排列 $b_1\,b_2\ \cdots\ b_{n-1}$;则 $b_1b_2\cdots b_{n-1}k$ 就是 $\{1,\cdots,n\}$ 的一个排列。

显然,通过这一构造,我们再次恰好得到 n 个元素的每一个排列一次。把 k 放在左边而不是在右边,或者把 k 放在任何其它的固定的位置,显然同样有效。

如果 p_n 是 n 个对象的排列的数目,上边这两种方法都表明 $p_n = np_{n-1}$;这就为我们提供了对于 $p_n = n(n-1)\cdots(1)$ 的两个进一步的证明,如同在等式(2)中已经确立的那样。

重要的数量 p_n 称做 n 的阶乘,它被写成

$$n! = 1 \cdot 2 \cdot \cdots \cdot n = \prod_{k=1}^{n} k \tag{4}$$

我们对于空积(1.2.3 小节)的约定为我们提供了值

$$0! = 1 \tag{5}$$

而且通过这一约定,对于所有正整数 n,就有基本恒等式

$$n! = (n-1)!\,n \tag{6}$$

成立。

在计算机著作中,阶乘经常出现,因此建议读者记住头几个阶乘的值:

$$0! = 1,\ 1! = 1,\ 2! = 2,\ 3! = 6,\ 4! = 24,\ 5! = 120$$

阶乘增长极其迅速:例如,1000! 是一个超过 2500 位的整数。

记住值 10! = 3 628 800 也是有用的;人们应当记住,10! 大约是 $3\frac{1}{2}$百万。在某种意义下,这个数表示能够计算的事物和不能计算的事物之间近似的分界线。如果一个算法要求对 10! 以上的情况进行测试,它可能要消耗太多的计算机时间因而是不实用的。另一方面,如果我们料定要测试 10! 种情况,而每一种情况,比如说,需要一毫秒的计算机时间,则整个运行将花费一个小时。当然,这些解释是很含糊的,但它们对于什么是计算上可行的直观概念来说,是有用的。

人们自然想知道 $n!$ 同数学上的其它数量有什么关系。如果不费尽气力地进行等式(4)所示的乘法,有无任何办法说出 1000! 有多大? 答案可从 James Stirling(斯特林)的名著 *Methodus Differentialis* (1730)的 137 页上找到;我们有

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n \tag{7}$$

这里符号"≈"表示"近似地等于",而 e 是在 1.2.2 小节中介绍的自然对数的底。我们将在 1.2.11.2 小节中证明斯特林的近似公式(7)。习题 24 给出一个不大精确的结果的简单证明。

作为使用这一个公式的一个例子,我们可以计算

$$40320 = 8! \approx 4\sqrt{\pi}\left(\frac{8}{\mathrm{e}}\right)^8 = 2^{26}\sqrt{\pi}\mathrm{e}^{-8} \approx 67108864 \cdot 1.77245 \cdot 0.00033546 \approx 39902$$

在这种情况下,误差大约是 1%;我们稍后将看到,相对误差近似于 $1/(12n)$。

除了由等式(7)给出的近似值外,我们还能稍微容易地得到把 $n!$ 分解成素数的精确值。事实上,素数 p 是 $n!$ 的重数为

$$\mu = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \cdots = \sum_{k>0} \left\lfloor \frac{n}{p^k} \right\rfloor \tag{8}$$

的一个因子。例如,如果 $n = 1000$ 和 $p = 3$,我们有

$$\mu = \left\lfloor \frac{1000}{3} \right\rfloor + \left\lfloor \frac{1000}{9} \right\rfloor + \left\lfloor \frac{1000}{27} \right\rfloor + \left\lfloor \frac{1000}{81} \right\rfloor + \left\lfloor \frac{1000}{243} \right\rfloor + \left\lfloor \frac{1000}{729} \right\rfloor =$$

$$333 + 111 + 37 + 12 + 4 + 1 = 498$$

所以 1000! 可由 3^{498} 所整除，但不能被 3^{499} 所整除。尽管公式(8)被写成无穷和，但实际上对于任何具体的 n 和 p 的值，它是有限的，因为后边所有项最终成为零。由习题 1.2.4-35 得出 $\lfloor n/p^{k+1}\rfloor = \lfloor\lfloor n/p^k\rfloor/p\rfloor$；这一事实简化了等式(8)中的计算，因为我们只须以 p 来除前一项的值而抛弃余数。

等式(8)是从这样一个事实得出的，$\lfloor n/p^k\rfloor$ 是在 $\{1,2,\cdots,n\}$ 当中为 p^k 之倍数者的个数。如果我们研究乘积(4)中的整数，其中为 p^j 所整除但不能为 p^{j+1} 所整除的任何整数恰好被计算了 j 次，一次在 $\lfloor n/p\rfloor$，一次在 $\lfloor n/p^2\rfloor,\cdots$，一次在 $\lfloor n/p^j\rfloor$，这就是 p 作为 $n!$ 的一个因子的所有出现情况。

产生了另外一个自然的问题：既然对于非负整数 n，我们定义了 $n!$，或许阶乘函数对于 n 的有理数值，甚至于实数值，也有意义。例如，什么是 $\left(\frac{1}{2}\right)!$。让我们通过引用“项”函数来说明这点：

$$n? = 1 + 2 + \cdots + n = \sum_{k=1}^{n} k \tag{9}$$

除了使用加来代替乘之外，它同阶乘函数是类似的。从等式 1.2.3-(15)我们已经知道这个算术级数的和：

$$n? = \frac{1}{2}n(n+1) \tag{10}$$

通过使用(10)来代替(9)，这提示了把“项”函数推广到任意 n 的一个好的方法。我们有 $\left(\frac{1}{2}\right)? = \frac{3}{8}$。

斯特林本人曾多次试图把 $n!$ 推广到非整数的 n。他把近似式(7)推广成无穷和，但不幸的是，对于任意的 n 值，这个和数不收敛；他的方法给出了极好的近似值，但它不能被推广来给出一个精确值。[关于这一有些异常情况的讨论，请参看 K. Knopp, *Theory and Application of Infinite Series*, 第 2 版 (Glasgow: Blackie, 1951), 518 ~ 520, 527, 534。]

通过注意到

$$n! = 1 + \left(1 - \frac{1}{1!}\right)n + \left(1 - \frac{1}{1!} + \frac{1}{2!}\right)n(n-1) + \left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!}\right)n(n-1)(n-2) + \cdots \tag{11}$$

斯特林再次进行了尝试(我们将在下一小节证明这个公式)。等式(11)中表面上的无穷求和实际上对于任何非负整数 n 是有限的；然而，它并不提供所希望的 $n!$ 的推广，因为除非当 n 是一个非负整数，否则这个无穷和不存在。(参见习题 16。)

斯特林仍不退缩，他发现了一个序列 $a_1, a_2, \cdots$，使得

$$\ln n! = a_1 n + a_2 n(n-1) + \cdots = \sum_{k\geq 0} a_{k+1} \prod_{0\leq j\leq k} (n-j) \tag{12}$$

尽管他有能力求出 $\left(\frac{1}{2}\right)! = \sqrt{\pi}/2$ 这个值，但却没有能力来证明，这个和对于 n 的所有

分数值定义了 $n!$。

大约在同一时期，欧拉研究了同一问题，而且他首先发现了适当的推广：

$$n! = \lim_{m\to\infty} \frac{m^n m!}{(n+1)(n+2)\cdots(n+m)} \tag{13}$$

欧拉于 1729 年 10 月 13 日写了一封信给 Christian Goldbach(哥德巴赫)，把这一思想告诉了他。他的公式对于除了负整数(当上式的分母变成 0 时)外的任意 n 值定义了 $n!$；在 n 是负整数的情况下 $n!$ 取作无穷大。习题 8 和 22 说明为什么等式(13)是一个合理的定义。

大约两个世纪之后，在 1900 年，C. Hermite 证明了，斯特林的思想(等式(12))实际上已经对于非整数的 n 成功地定义了 $n!$，因此，事实上欧拉和斯特林的推广是相同的。

历史上，对于阶乘使用过许多记号，欧拉实际上写成$[n]$，高斯写成$\Pi\, n$，而在英国和意大利符号$\lfloor\underline{n}$ 和 $\underline{n}\rfloor$曾经很流行。今天普遍采用的符号 $n!$ (当 n 是整数时)是由相对不大出名的数学家 Christian Kramp 在一本代数教科书[*Élémens d'Arithmétique Universelle* (Cologne: 1808)]中引进的。

然而，当 n 不是整数时，符号 $n!$ 不大常用，而习惯上使用勒让德给出的一个符号：

$$n! = \Gamma(n+1) = n\Gamma(n) \tag{14}$$

这个函数 $\Gamma(x)$叫做伽玛(gamma)函数，而且由等式(13)，我们有定义

$$\Gamma(x) = \frac{x!}{x} = \lim_{m\to\infty} \frac{m^x m!}{x(x+1)(x+2)\cdots(x+m)} \tag{15}$$

图 7 中给出了 $\Gamma(x)$的图形。

等式(13)和(15)既定义了实数的也定义了复数的阶乘和伽玛函数；但当想像一个变量既有实部又有虚部时，代替使用 n 或 x，我们一般使用字母 z。阶乘和伽玛函数不仅通过规则 $z! = \Gamma(z+1)$相关联，而且也通过

$$(-z)!\Gamma(z) = \frac{\pi}{\sin \pi z} \tag{16}$$

相互关联，当 z 不是整数时，这个式子成立(见习题 23)。

尽管当 z 为零或负整数时，$\Gamma(z)$无穷大，但函数 $1/\Gamma(z)$对于所有复数 z 都有明确定义。(见习题 1.2.7-24。)伽玛函数的高级应用经常利用由 Hermann Hankel 给出的重要的曲线积分公式

$$\frac{1}{\Gamma(z)} = \frac{1}{2\pi i}\oint \frac{e^t\,dt}{t^z} \tag{17}$$

复数积分的路径从 $-\infty$开始，然后以逆时针方向围绕圆心并返回到 $-\infty$处。[*Zeitschrift für Math. und Physik* **9** (1864), 1～21。]

离散数学的许多公式涉及称做阶乘幂的类似阶乘的积。当 k 是正整数时，定义数

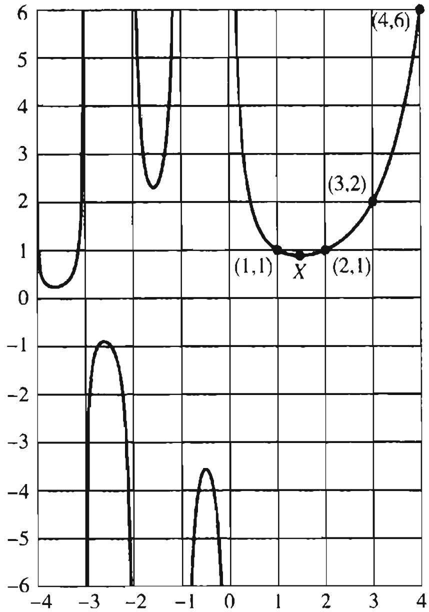


图 7　函数 $\Gamma(x)=(x-1)!$

在 X 处的极小值的坐标为(1.46163 21449 68362 34126 26595,0.88560 31944 10888 70027 88159)。

量 $x^{\underline{k}}$和 $x^{\overline{k}}$(读作"x 降到 k"和"x 升到 k")如下：

$$x^{\underline{k}} = x(x-1)\cdots(x-k+1) = \prod_{j=0}^{k-1}(x-j) \tag{18}$$

$$x^{\overline{k}} = x(x+1)\cdots(x+k-1) = \prod_{j=0}^{k-1}(x+j) \tag{19}$$

因此,(2)的数 p_{nk}仅仅是 $n^{\underline{k}}$。注意,我们有

$$x^{\overline{k}} = (x+k-1)^{\underline{k}} = (-1)^k(-x)^{\underline{k}} \tag{20}$$

一般的公式

$$x^{\underline{k}} = \frac{x!}{(x-k)!},\quad x^{\overline{k}} = \frac{\Gamma(x+k)}{\Gamma(x)} \tag{21}$$

可以用来定义 k 的其它值的阶乘幂。[记号 $x^{\overline{k}}$是由 A.Capelli 给出的,*Giornale di Matematiche di Battaglini* **31** (1893),291 ~ 313。]

关于阶乘由斯特林时代直到今天的有趣历史,见 P. J. Davis 的一篇文章"Leonhard Euler's integral:A historical profile of the gamma function", *AMM* **66** (1959),849 ~ 869。也请参看 J.Dutka, *Archive for History of Exact Sciences* **31** (1984), 15 ~ 34。

习 题

1. [*00*] 有多少种办法来洗 52 张纸牌?

2. [*10*] 在等式(2)的记号下,证明 $p_{n(n-1)} = p_{nn}$,并说明为什么出现这一情况。

3. [*10*] 从排列 3 1 2 4,分别使用方法 1 和方法 2,将构造出{1,2,3,4,5}的什么排列?

▶ 4. [*13*] 给定 $\log_{10} 1000! = 2567.60464\cdots$这一事实,精确地确定在数 1000! 中有多少位数出现? 最高有效位数字是什么? 最低有效位数字是什么?

5. [*15*] 使用以下斯特林近似公式更精确的形式

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \frac{1}{12n}\right)$$

估计 8!。

▶ 6. [*17*] 利用等式(8),把 20! 写成素因子的一个乘积。

7. [*M10*] 证明等式(10)中的"广义的项"函数,对于所有实数 x,满足 $x? = x + (x-1)?$。

8. [*HM15*] 证明当 n 是非负整数时,等式(13)中极限确实等于 $n!$。

9. [*M10*] 给定$\left(\frac{1}{2}\right)! = \sqrt{\pi}/2$,确定 $\Gamma\left(\frac{1}{2}\right)$ 和 $\Gamma\left(-\frac{1}{2}\right)$ 的值。

▶ 10. [*HM20*] 等式 $\Gamma(x+1) = x\Gamma(x)$是否对于所有实数 x 都成立?(见习题 7。)

11. [*M15*] 设 n 在二进制系统中的表示是 $n = 2^{e_1} + 2^{e_2} + \cdots + 2^{e_r}$,其中 $e_1 > e_2 > \cdots > e_r \geqslant 0$,说明 $n!$ 可由 2^{n-r}但不能由 2^{n-r+1}所整除。

▶ 12. [*M22*](A. Legendre, 1808) 推广上一习题的结果,设 p 是素数,且设 n 在 p 进系统下的表示是 $n = a_k p^k + a_{k-1} p^{k-1} + \cdots + a_1 p + a_0$。以涉及 n, p 和诸 a 的一个简单公式表示等式(8)中的数 μ。

13. [*M23*] (Wilson 定理,实际上是由莱布尼茨给出的,1682) 如果 p 是素数,则$(p-1)! \bmod p = p-1$。通过把$\{1,2,\cdots,p-1\}$当中,其模 p 乘积为 1 的数结成对证明这一点。

▶ 14. [*M28*] (L. Stickelberger, 1890) 在习题 12 的记号下,对于任何正整数 n,借助于 p 进表示,我们可以确定 $n! \bmod p$,由此推广 Wilson 定理。事实上,证明 $n!/p^\mu \equiv (-1)^\mu a_0!\, a_1! \cdots a_k!$ (modulo p)。

15. [*HM15*] 通过和行列式相同的展开,来定义一个方阵的**积和式**(permanent),不同的是,行列式的项交替地带正号和负号,而积和式的每一项都带正号。因此

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

的积和式是 $aei + bfg + cdh + gec + hfa + idb$,今问

$$\begin{pmatrix} 1\times 1 & 1\times 2 & \cdots & 1\times n \\ 2\times 1 & 2\times 2 & \cdots & 2\times n \\ \vdots & \vdots & \vdots & \vdots \\ n\times 1 & n\times 2 & \cdots & n\times n \end{pmatrix}?$$

的积和式是什么?

16. [*HM15*] 证明等式(11)中的无穷和不收敛,除非 n 是非负整数。

17. [*HM20*] 证明:如果 $\alpha_1+\cdots+\alpha_k=\beta_1+\cdots+\beta_k$,且如果诸 β 不是负整数,则无穷乘积

$$\prod_{n\geq 1}\frac{(n+\alpha_1)\cdots(n+\alpha_k)}{(n+\beta_1)\cdots(n+\beta_k)}$$

等于 $\Gamma(1+\beta_1)\cdots\Gamma(1+\beta_k)/\Gamma(1+\alpha_1)\cdots\Gamma(1+\alpha_k)$。

18. [*M20*] 假设 $\pi/2=\frac{2}{1}\cdot\frac{2}{3}\cdot\frac{4}{3}\cdot\frac{4}{5}\cdot\frac{6}{5}\cdot\frac{6}{7}\cdot\cdots$(这是由 J. Wallis 于 1655 年得到的"Wallis 乘积",我们将在习题 1.2.6-43 中证明之)。试用上题的结果证明$\left(\frac{1}{2}\right)!\ =\sqrt{\pi}/2$。

19. [*HM22*] 用 $\Gamma_m(x)$表示在等式(15)中"$\lim_{m\to\infty}$"之后出现的量,证明,如果 $x>0$,

$$\Gamma_m(x)=\int_0^m\left(1-\frac{t}{m}\right)^m t^{x-1}\mathrm{d}t=m^x\int_0^1(1-t)^m t^{x-1}\mathrm{d}t$$

20. [*HM21*] 利用事实如果 $0\leq t\leq m$,则 $0\leq e^{-t}-(1-t/m)^m\leq t^2e^{-t}/m$,以及上一题的结果,证明,如果 $x>0$,则 $\Gamma(x)=\int_0^\infty e^{-t}t^{x-1}\mathrm{d}t$。

21. [*HM25*] (L. F. A. Arbogast, 1800) 设 $D_x^k u$ 表示函数 u 关于 x 的第 k 阶导数,链规则指出 $D_x^1 w=D_u^1 wD_x^1 u$。如果我们把这应用于二阶导数,发现 $D_x^2 w=D_u^2 w(D_x^1 u)^2+D_u^1 wD_x^2 u$。证明一般公式是

$$D_x^n w=\sum_{j=0}^{n}\sum_{\substack{k_1+k_2+\cdots+k_n=j\\ k_1+2k_2+\cdots+nk_n=n\\ k_1,k_2,\cdots,k_n\geq 0}}D_u^j w\frac{n!}{k_1!(1!)^{k_1}\cdots k_n!(n!)^{k_n}}(D_x^1 u)^{k_1}\cdots(D_x^n u)^{k_n}$$

▶ 22. [*HM20*] 试把你自己设想成处于欧拉的地位,来寻找把 $n!$ 推广到非整数的 n 值的方法。由于$(n+\frac{1}{2})!/n!$ 乘$\left((n+\frac{1}{2})+\frac{1}{2}\right)!\Big/(n+\frac{1}{2})!$ 等于$(n+1)!/n!\ =n+1$,因此看起来$(n+\frac{1}{2})!/n!$ 应近似于$\sqrt{n}$是自然的。类似地,$(n+\frac{1}{3})!/n!$ 应是$\approx\sqrt[3]{n}$。试想出当 n 趋于无穷大时关于比值$(n+x)!/n!$ 的一个假设。当 x 为整数时,你的假设是否正确?如果 x 不是整数,关于 $x!$ 的相应值,这个假设说明什么呢?

23. [*HM20*] 给定 $\pi z\prod_{n=1}^{\infty}(1-z^2/n^2)=\sin\pi z$,证明(16)。

▶ 24. [*HM21*] 当整数 $n\geq 1$ 时,证明简单的不等式$\frac{n^n}{e^{n-1}}\leq n!\leq\frac{n^{n+1}}{e^{n-1}}$。

[提示:对于所有实数 x,$1+x\leq e^x$;因此$(k+1)/k\leq e^{1/k}\leq k/(k-1)$。]

25. [*M20*] 阶乘幂是否满足类似于通常的指数律 $x^{m+n}=x^mx^n$ 的一个定律?

1.2.6　二项式系数

从 n 个对象中一次取 k 个的组合是不考虑顺序,从 n 个对象的一个集合中取 k 个不同的元素的可能的选择数。从五个对象$\{a,b,c,d,e\}$一次取三个的组合是

$$abc,\ abd,\ abe,\ acd,\ ace,\ ade,\ bcd,\ bce,\ bde,\ cde \qquad (1)$$

计算 n 个对象的 k 个组合的总数是一件简单的事:上一小节的等式(2)告诉我们对于一个排列来说选择头 k 个对象有 $n(n-1)\cdots(n-k+1)$ 种方法,而且在这些排列当中,每一种 k 组合恰出现 $k!$ 次,因为每一组合都出现在它的所有排列中。因此组合的总数,我们记之为 $\binom{n}{k}$,是

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots(1)} \qquad (2)$$

例如

$$\binom{5}{3} = \frac{5\cdot 4\cdot 3}{3\cdot 2\cdot 1} = 10$$

它就是我们在(1)中找出的组合的数目。

量 $\binom{n}{k}$,读作"从 n 选 k",称为二项式系数;这些数有格外多的应用。它们可能是进入到算法分析的最重要的量,因此读者应当熟悉它们。

即便 n 不是一个整数,等式(2)也可用来定义 $\binom{n}{k}$。更精确地说,我们对于所有实数 r 和所有整数 k 定义符号 $\binom{r}{k}$ 如下:

$$\binom{r}{k} = \frac{r(r-1)\cdots(r-k+1)}{k(k-1)\cdots(1)} = \frac{r^{\underline{k}}}{k!} = \prod_{j=1}^{k} \frac{r+1-j}{j}, \qquad \text{整数 } k \geqslant 0$$

$$\binom{r}{k} = 0, \qquad \text{整数 } k < 0 \qquad (3)$$

在特殊情况下,我们有

$$\binom{r}{0} = 1,\quad \binom{r}{1} = r,\quad \binom{r}{2} = \frac{r(r-1)}{2} \qquad (4)$$

表 1 给出对于小的整数 r 和 k 的值二项式系数的值;对于 $0 \leqslant r \leqslant 4$ 条件下的值都应当记住。

二项式系数有一段漫长而有趣的历史。表 1 之所以叫做帕斯卡三角,是因为它出现于 1653 年的 Blaise Pascal 的著作 *Traité du Triangle Arithmétique* 中。这本专著是有历史意义的,因为它是关于概率论的最早的论著之一。但是帕斯卡不是二项式系数的发明者(那时二项式系数在欧洲已为人所熟知)。表 1 也出现在 1303 年中国数学家朱世杰

表 1　二项式系数表(帕斯卡三角)*

r	$\binom{r}{0}$	$\binom{r}{1}$	$\binom{r}{2}$	$\binom{r}{3}$	$\binom{r}{4}$	$\binom{r}{5}$	$\binom{r}{6}$	$\binom{r}{7}$	$\binom{r}{8}$	$\binom{r}{9}$
0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
2	1	2	1	0	0	0	0	0	0	0
3	1	3	3	1	0	0	0	0	0	0
4	1	4	6	4	1	0	0	0	0	0
5	1	5	10	10	5	1	0	0	0	0
6	1	6	15	20	15	6	1	0	0	0
7	1	7	21	35	35	21	7	1	0	0
8	1	8	28	56	70	56	28	8	1	0
9	1	9	36	84	126	126	84	36	9	1

(Chu Shih-Chieh)的专著《四元玉鉴》中,书中说二项式系数是一项老发明。杨辉于 1261 年把它们归功于贾宪,但贾宪的著作(约 1000 年)已轶失。已知的关于二项式系数最早的讨论,可追溯到 10 世纪的一篇由 Halāyuddha 所著的,关于一位印度古典作家 Piṅgala 的作品*Chandahśāstra* 的评论。[见 G. Chakravarti, *Bull. Calcutta. Math. Soc.* **24**(1932),79 ~ 88。]大约 1150 年,印度数学家 Bhāskarācārya 在他的著作*Līlāvatī* 第 6 部分第 4 章中,给出了关于二项式系数的非常清楚的说明。对于小的 k 值,知道得更早;它们以一个几何解释(见图 8)出现于希腊和罗马的一些著作中。记号$\binom{r}{k}$是由 Andreas von Ettingshausen 在他所写的书*Die combinatorische Analysis* (Vienna: 1826)中引进的。

读者大概已经看出表 1 中若干有趣的样式,二项式系数字面上满足数以千计的恒等式,而且若干世纪以来,它们令人惊异的性质已经陆续地被剖析。事实上,现有的关系式已经是如此之多,以致当某人发现一个新的恒等式时,除了发现者本人之外,再没有多少人为之感到激动。为了处理出现在算法分析中的公式,必须有处理二项式系数的工具。因此在本小节里试图以简明的方式来说明怎样对这些数进行演算。马克·吐温(Mark Twain)首先尝试把所有的笑话归结成一打左右的非常原始的类型(农夫的女儿,岳母,等等);而我们则尝试把数以千计的恒等式归结成基本运算的一个小集合。通过这些基本运算,就能解决我们将面对的涉及二项式系数的几乎每一个问题。

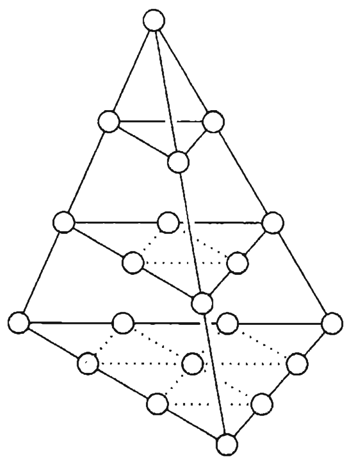


图 8　$\binom{n+2}{3}$, $n=4$ 的几何解释

* 实为杨辉三角。——译者注

在大多数应用中，出现在$\binom{r}{k}$中的两个数 r 和 k 都是整数，而且我们将描述的某些技术仅仅在这样的情况下是适用的。所以我们将在每一个编了号的等式的右边，都仔细地列出对于出现的变量的限制。例如，等式(3)提出 k 是整数的要求，而对 r 不加限制。具有最少限制的恒等式是最有用的。

现在就让我们来研究对二项式系数进行运算的基本技术。

A. 用阶乘来表示 由等式(3)我们立即有

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}, \qquad \text{整数 } n \geqslant \text{整数 } k \geqslant 0 \tag{5}$$

这允许我们把阶乘的组合表示成二项式，以及反过来，把二项式系数表示成阶乘的组合。

B. 对称条件 由等式(3)和(5)，我们有

$$\binom{n}{k} = \binom{n}{n-k}, \qquad \text{整数 } n \geqslant 0, \text{整数 } k \tag{6}$$

这个公式对于所有整数 k 都成立。*当 k 为负或大于 n 时，二项式系数为零*（假设 n 是一个非负整数）。

C. 移进和移出括弧 由定义(3)，我们有

$$\binom{r}{k} = \frac{r}{k}\binom{r-1}{k-1}, \qquad \text{整数 } k \neq 0 \tag{7}$$

这个公式对于把一个二项式系数同一个表达式的其它部分组合在一起非常有用。通过初等变换，我们有规则

$$k\binom{r}{k} = r\binom{r-1}{k-1}, \qquad \frac{1}{r}\binom{r}{k} = \frac{1}{k}\binom{r-1}{k-1}$$

上边两个规则中的头一个对所有整数 k 都成立，而第二个只要不出现以 0 作为除数的情况也都成立，我们还有一个类似的关系

$$\binom{r}{k} = \frac{r}{r-k}\binom{r-1}{k}, \qquad \text{整数 } k \neq r \tag{8}$$

现在让我们通过交替地使用等式(6)和(7)证明等式(8)，来说明这些变换：

$$\binom{r}{k} = \binom{r}{r-k} = \frac{r}{r-k}\binom{r-1}{r-1-k} = \frac{r}{r-k}\binom{r-1}{k}$$

[注：这个推导仅当 r 是一个不等于 k 的正整数时才正确，因为这是在等式(6)和(7)中包含的限制；但(8)断言对于任意 $r \neq k$ 都正确，这可以以一个简单但是重要的方式证明：我们已经验证了，对于无穷多的 r 值，

$$r\binom{r-1}{k}=(r-k)\binom{r}{k}$$

这个等式的两边都是 r 的多项式,一个 n 次的非 0 多项式至多能有 n 个不同的零点;所以(通过减法),*如果小于等于 n 次的两个多项式在 $n+1$ 个或更多个不同的点上相等,则这两个多项式恒等*。这一原理可以用来把对于整数成立的许多等式的正确性推广到所有实数的情形。]

D. 加法公式　基本关系式

$$\binom{r}{k}=\binom{r-1}{k}+\binom{r-1}{k-1},\qquad \text{整数 } k \tag{9}$$

在表 1 中显然成立(每个值都是上一个与这上一个左边那一个的两个值之和),而且从等式(3)我们可以一般地验证它。或者,等式(7)和(8)告诉我们

$$r\binom{r-1}{k}+r\binom{r-1}{k-1}=(r-k)\binom{r}{k}+k\binom{r}{k}=r\binom{r}{k}$$

当 r 是整数时,如果要得到通过对 r 使用归纳法的证明,则等式(9)通常是有用的。

E. 求和公式　反复地应用等式(9),我们得到

$$\binom{r}{k}=\binom{r-1}{k}+\binom{r-1}{k-1}=\binom{r-1}{k}+\binom{r-2}{k-1}+\binom{r-2}{k-2}=\cdots$$

或者

$$\binom{r}{k}=\binom{r-1}{k-1}+\binom{r-1}{k}=\binom{r-1}{k-1}+\binom{r-2}{k-1}+\binom{r-2}{k}=\cdots$$

因此就为我们导出了两个重要的求和公式,它们可表达如下:

$$\sum_{k=0}^{n}\binom{r+k}{k}=\binom{r}{0}+\binom{r+1}{1}+\cdots+\binom{r+n}{n}=\binom{r+n+1}{n},\quad \text{整数 } n\geqslant 0 \tag{10}$$

$$\sum_{k=0}^{n}\binom{k}{m}=\binom{0}{m}+\binom{1}{m}+\cdots+\binom{n}{m}=\binom{n+1}{m+1},\quad \text{整数 } m\geqslant 0,\text{整数 } n\geqslant 0 \tag{11}$$

通过对 n 使用归纳法能很容易地证明等式(11)。但是看一看,通过两次应用等式(6),它也可以由等式(10)导出,也是有趣的:

$$\sum_{0\leqslant k\leqslant n}\binom{k}{m}=\sum_{0\leqslant m+k\leqslant n}\binom{m+k}{m}=\sum_{-m\leqslant k<0}\binom{m+k}{m}+\sum_{0\leqslant k\leqslant n-m}\binom{m+k}{k}=$$

$$0+\binom{m+(n-m)+1}{n-m}=\binom{n+1}{m+1}$$

这里假定了 $n\geqslant m$。如果 $n<m$,则等式(11)是显然的。

在应用中,等式(11)出现得非常频繁。事实上,在前边几个小节里我们已经推导出了它的一些特殊情况。例如,当 $m=1$ 时,我们有熟知的情况,即算术级数之和:

$$\binom{0}{1}+\binom{1}{1}+\cdots+\binom{n}{1}=0+1+\cdots+n=\binom{n+1}{2}=\frac{(n+1)n}{2}$$

假设我们要求和数 $1^2+2^2+\cdots+n^2$ 的一个简单公式。这可以通过注意到 $k^2=2\binom{k}{2}+\binom{k}{1}$ 而得;因此

$$\sum_{k=0}^{n}k^2=\sum_{k=0}^{n}\left(2\binom{k}{2}+\binom{k}{1}\right)=2\binom{n+1}{3}+\binom{n+1}{2}$$

而且借助于二项式系数得到的这个答案,如果愿意,还可以变回多项式表示如下:

$$1^2+2^2+\cdots+n^2=2\frac{(n+1)n(n-1)}{6}+\frac{(n+1)n}{2}=\frac{1}{3}n(n+\frac{1}{2})(n+1)\quad(12)$$

以类似的方式还可得到和数 $1^3+2^3+\cdots+n^3$;对于适当选择的系数 $b_0,\cdots,b_m$,任何多项式 $a_0+a_1k+a_2k^2+\cdots+a_mk^m$ 都可表达成 $b_0\binom{k}{0}+b_1\binom{k}{1}+\cdots+b_m\binom{k}{m}$。后边我们还要返回到这个论题上来。

F. 二项式定理 当然,二项式定理是我们的主要工具之一:

$$(x+y)^r=\sum_k\binom{r}{k}x^ky^{r-k},\qquad 整数\ r\geqslant 0\qquad(13)$$

例如,$(x+y)^4=x^4+4x^3y+6x^2y^2+4xy^3+y^4$。(这就是我们称 $\binom{r}{k}$ 为“二项式系数”的原因。)

有必要注意,在等式(13)中我们写的是 $\sum_k$,而不是以前常写的 $\sum_{k=0}^r$。如果对 k 不加限制,我们便是对所有整数 $-\infty<k<+\infty$ 求和;但在这个情况下,两个记号完全等价,因为当 $k<0$ 或 $k>r$ 时等式(13)中的项为零。较简单的形式 $\sum_k$ 是可取的,因为当求和的条件较简单时,对于求和的所有操作也是较简单些的。如果我们无须记住求和的下限和/或上限,我们便可节省大量的繁琐的努力,因而只要可能就应使限定保持为不确定的。我们的记号还有另外一个优点:如果 r 不是一个非负整数,等式(13)就变成一个无穷和,而微积分的二项式定理指出,如果 $|x/y|<1$,等式(13)对于所有 r 都正确。

还应指出，公式(13)给出

$$0^0 = 1 \tag{14}$$

我们将始终使用这一约定。

等式(13)中的特殊情况 $y=1$ 也是重要的，因此我们特别地提出来：

$$\sum_k \binom{r}{k} x^k = (1+x)^r, \qquad \text{整数 } r \geqslant 0 \text{ 或 } |x| < 1 \tag{15}$$

二项式定理的发现是由牛顿于 1676 年 6 月 13 日和 1676 年 10 月 24 日在给 Oldenburg 的信中宣布的。[见 D. Struik, *Source Book in Mathematics* (Harvard Univ. Press, 1969), 284～291。]但他显然没有给出这一公式的实际证明；当时人们还没有充分认识到严格证明的必要性。第一个证明尝试是由欧拉于 1774 年给出的，然而该证明也不完全。最后，高斯于 1812 年给出了头一个真正的证明。事实上，高斯的工作代表了关于无穷和的第一个完满的证明。

19 世纪初，N. H. Abel(阿贝尔)发现了关于二项式公式(13)的一个惊人的推广：

$$(x+y)^n = \sum_k \binom{n}{k} x(x-kz)^{k-1}(y+kz)^{n-k}, \qquad \text{整数 } n \geqslant 0, x \neq 0 \tag{16}$$

这是三个变量 x, y 和 z 的一个恒等式(参看习题 50～52)。阿贝尔在 A. L. Crelle 很快就闻名的 *Journal für die reine und angewandte Mathematik* 第 1 卷(1826)，159～160 页上发表和证明了这一公式。有趣的是阿贝尔在同一卷上还同时发表了好多篇论文，其中包括关于五次或以上代数方程用根号的不可解性及关于二项式定理的论文。有关等式(16)的一些参考文献，请参见 H. W. Gould, *AMM* **69** (1962), 572。

G. 把上标取负　由定义(3)，当分子的每一项取负号时，立即得出以下的基本恒等式

$$\binom{r}{k} = (-1)^k \binom{k-r-1}{k}, \qquad \text{整数 } k \tag{17}$$

这常常是有用的对于上标的变换。

等式(17)的一个容易的推论是求和公式

$$\sum_{k \leqslant n} \binom{r}{k}(-1)^k = \binom{r}{0} - \binom{r}{1} + \cdots + (-1)^n \binom{r}{n} = (-1)^n \binom{r-1}{n}, \quad \text{整数 } n \tag{18}$$

这个恒等式可以通过使用等式(9)用归纳法来证明，但我们也可直接使用等式(17)和等式(10)来证明：

$$\sum_{k \leqslant n} \binom{r}{k}(-1)^k = \sum_{k \leqslant n} \binom{k-r-1}{k} = \binom{-r+n}{n} = (-1)^n \binom{r-1}{n}$$

当 r 是整数时，可以给出等式(17)的另一个重要的应用：

$$\binom{n}{m} = (-1)^{n-m}\binom{-(m+1)}{n-m}, \qquad \text{整数 } n \geq 0, \text{整数 } m \tag{19}$$

(在等式(17)中，置 $r = n$ 和 $k = n - m$，并使用(6)。)我们已把 n 从上边的位置移到下边的位置来了。

H. 简化乘积　当出现二项式系数的乘积时，通过展开成阶乘，而后再使用等式(5)，它们通常可以以若干不同的方式重新表示。例如，

$$\binom{r}{m}\binom{m}{k} = \binom{r}{k}\binom{r-k}{m-k}, \qquad \text{整数 } m, \text{整数 } k \tag{20}$$

只要 $r \geqslant m$ 是整数(参见等式(8)之后的论述)以及 $0 \leqslant k \leqslant m$ 即足以证明。于是

$$\binom{r}{m}\binom{m}{k} = \frac{r!\,m!}{m!\,(r-m)!\,k!\,(m-k)!} =$$

$$\frac{r!\,(r-k)!}{k!\,(r-k)!\,(m-k)!\,(r-m)!} = \binom{r}{k}\binom{r-k}{m-k}$$

当一个下标(即 m)在上边的位置和下边的位置同时出现，而我们希望它只出现在一个位置而不是在两个位置时，等式(20)是非常有用的。注意，等式(7)是当 $k = 1$ 时等式(20)的特殊情况。

I. 乘积的和　为了完成对二项式系数的整套处理方法的讨论，我们给出下列非常一般的恒等式。它们的证明留作本小节末尾的习题。这些公式说明怎样对两个二项式系数的乘积进行求和，同时考虑到游动变量 k 可能出现的各种位置：

$$\sum_k \binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}, \qquad \text{整数 } n \tag{21}$$

$$\sum_k \binom{r}{m+k}\binom{s}{n+k} = \binom{r+s}{r-m+n}, \qquad \text{整数 } m, \text{整数 } n, \text{整数 } r \geqslant 0 \tag{22}$$

$$\sum_k \binom{r}{k}\binom{s+k}{n}(-1)^{r-k} = \binom{s}{n-r}, \qquad \text{整数 } n, \text{整数 } r \geqslant 0 \tag{23}$$

$$\sum_{k=0}^{r} \binom{r-k}{m}\binom{s}{k-t}(-1)^{k-t} = \binom{r-t-s}{r-t-m}, \qquad \text{整数 } t \geqslant 0, \text{整数 } r \geqslant 0, \text{整数 } m \geqslant 0 \tag{24}$$

$$\sum_{k=0}^{r}\binom{r-k}{m}\binom{s+k}{n}=\binom{r+s+1}{m+n+1},\qquad \text{整数 } n\geqslant \text{整数 } s\geqslant 0,\ \text{整数 } m\geqslant 0,\text{整数 } r\geqslant 0 \tag{25}$$

$$\sum_{k\geqslant 0}\binom{r-tk}{k}\binom{s-t(n-k)}{n-k}\frac{r}{r-tk}=\binom{r+s-tn}{n},\qquad \text{整数 } n \tag{26}$$

在这些等式中,等式(21)是至今最重要的一个,因而应当记住它。记住它的一个方法是把右边解释为从 r 个男人和 s 个女人中来选择 n 个人;而在左边中的每一项是从男人中选择出 k 个人而从女人中选择出 $n-k$ 个人来的方法数。等式(21)通常叫做范德蒙德综合式,因为 A. Vandermonde 在*Mém. Acad. Roy. Sciences* (Paris, 1772), Part 1, 489 ~ 498 上发表了它。但是它在前边提到的朱世杰 1303 年的论著中就已经出现了[参见 J. Needham(李约瑟), *Science and Civilisation in China* **3** (Cambridge University Press, 1959), 138 ~ 139。]

如果在等式(26)中, $r=tk$,则我们可以通过删去分子中的一个因子而避免零分母;因此,等式(26)是变量 r,s,t 的一个多项式恒等式。显然,等式(21)是 $t=0$ 时等式(26)的特殊情况。

我们还应指出等式(23)和(25)的一个不明显的用法:以左边更复杂的表达式来代替右边简单的二项式系数,交换求和的顺序,并进行简化,通常是有帮助的。我们可以认为左边是 $\binom{s}{n+a}$ 借助于 $\binom{s+k}{n}$ 的展开。公式(23)用于负的 a,而公式(25)用于正的 a。

这就完成了我们对于二项式系数技术的研究。建议读者对它认真地学习,特别是等式(5),(6),(7),(9),(13),(17),(20)和(21)——用你所喜欢的方式把它们重点标出来。

有了所有这些方法在手,我们应当有能力至少以三种不同的方法来解决出现的几乎任何问题。以下的例子就来说明这些技术。

例 1　当 r 是正整数时, $\sum_k\binom{r}{k}\binom{s}{k}k$ 的值是什么?

解:对于处置外边的 k 说来,公式(7)是有用的:

$$\sum_k\binom{r}{k}\binom{s}{k}k=\sum_k\binom{r}{k}\binom{s-1}{k-1}s=s\sum_k\binom{r}{k}\binom{s-1}{k-1}$$

现在对于 $m=0$ 且 $n=-1$,公式(22)成立。因此答案是

$$\sum_k\binom{r}{k}\binom{s}{k}k=\binom{r+s-1}{r-1}s,\qquad \text{整数 } r\geqslant 0$$

例 2　如果 n 是非负整数, $\sum_k\binom{n+k}{2k}\binom{2k}{k}\frac{(-1)^k}{k+1}$ 的值是 什么?

解：这一问题更棘手；求和的下标 k 出现于六处！首先我们应用等式(20)，因而得到

$$\sum_k \binom{n+k}{k}\binom{n}{k}\frac{(-1)^k}{k+1}$$

我们现在可以轻松一点，因为原来的公式的一些威胁性的特征已经消失。下一步应是明显的；我们以类似于在例 1 中所使用技术的方式应用等式(7)：

$$\sum_k \binom{n+k}{k}\binom{n+1}{k+1}\frac{(-1)^k}{n+1} \tag{27}$$

好，另一个 k 又消失了。现在有两个同样有希望的解决路线。假定 $k \geqslant 0$，我们可以以 $\binom{n+k}{n}$ 代替 $\binom{n+k}{k}$，并且通过等式(23)来计算这个和：

$$\sum_{k\geqslant 0}\binom{n+k}{n}\binom{n+1}{k+1}\frac{(-1)^k}{n+1} =$$

$$-\frac{1}{n+1}\sum_{k\geqslant 1}\binom{n-1+k}{n}\binom{n+1}{k}(-1)^k =$$

$$-\frac{1}{n+1}\sum_{k\geqslant 0}\binom{n-1+k}{n}\binom{n+1}{k}(-1)^k + \frac{1}{n+1}\binom{n-1}{n} =$$

$$-\frac{1}{n+1}(-1)^{n+1}\binom{n-1}{-1} + \frac{1}{n+1}\binom{n-1}{n} =$$

$$\frac{1}{n+1}\binom{n-1}{n}$$

二项式系数 $\binom{n-1}{n}$ 等于零，除非 $n=0$ 时，它等于 1。因此我们可以方便地写出答案，使用 Iverson 约定（等式 1.2.3-(16)）为 $[n=0]$，或者使用克罗内克符号（等式 1.2.3-(19)）为 δ_{n0}。

从等式(27)进行的另一个办法是使用等式(17)，得到

$$\sum_k \binom{-(n+1)}{k}\binom{n+1}{k+1}\frac{1}{n+1}$$

我们现在可以应用等式(22)，它产生和

$$\binom{n+1-(n+1)}{n+1-1+0}\frac{1}{n+1}=\binom{0}{n}\frac{1}{n+1}$$

再一次我们导出答案

$$\sum_k \binom{n+k}{2k}\binom{2k}{k}\frac{(-1)^k}{k+1}=\delta_{n0},\qquad \text{整数 } n\geqslant 0 \tag{28}$$

例 3　对于正整数 m 和 n，$\sum\binom{n+k}{m+2k}\binom{2k}{k}\frac{(-1)^k}{k+1}$ 的值是什么？

解：如果 m 为零，我们将有和在例 2 中一样的公式。然而，m 的存在意味着我们甚至不能使用以前求解的方法，因为在那里的头一步是使用等式(20)——现在它不再可用了。在这种情况下，值得通过形如$\binom{x+k}{2k}$的项之和来代替不想要的$\binom{n+k}{m+2k}$，而使事情更复杂化，因为那样一来我们的问题就将变成我们知道如何求解的问题之和。因此，对于 $r=n+k-1$，$m=2k$，$s=0$，$n=m-1$，使用等式(25)，我们有

$$\sum_k\sum_{0\leqslant j\leqslant n+k-1}\binom{n+k-1-j}{2k}\binom{2k}{k}\binom{j}{m-1}\frac{(-1)^k}{k+1} \tag{29}$$

我们希望首先对 k 进行求和；但交换求和的顺序要求我们对于大于等于 0 和大于等于 $j-n+1$的 k 值进行求和。不幸的是，后一个条件引起问题，因为当 $j\geqslant n$ 时我们不知道所希望的和了。然而，通过观察到当 $n\leqslant j\leqslant n+k-1$ 时(29)的项变为零，即可使我们解救危局。这个条件意味着 $k\geqslant 1$；因此 $0\leqslant n+k-1-j\leqslant k-1<2k$；而且(29)中的头一个二项式系数将消失。因此我们可以以 $0\leqslant j<n$ 来代替第二个求和的条件，因而求和的交换就是小事一桩了。通过等式(28)对 k 求和现在给出

$$\sum_{0\leqslant j<n}\binom{j}{m-1}\delta_{(n-1-j)0}$$

因此除了当 $j=n-1$ 外，所有项皆为零。因此最后的答案是

$$\binom{n-1}{m-1}$$

这个问题的解是相当复杂的，但是实际上并不神秘；每一步都有很好的理由。应仔细研究这个推导，因为它展示了处理等式条件的某些巧妙的策略。然而，实际上还有更好的办法来解决这个问题，这留给读者：想出转换给定的和的一个办法，使得可应用等式(26)(见习题 30)。

例 4　证明

$$\sum_k A_k(r,t)A_{n-k}(s,t)=A_n(r+s,t),\qquad \text{整数 } n\geqslant 0 \tag{30}$$

其中 $A_n(x,t)$是 x 的第 n 次多项式，它满足

$$A_n(x,t)=\binom{x-nt}{n}\frac{x}{x-nt},\qquad x\neq nt$$

解：可以假定对于 $0\leqslant k\leqslant n$，$r\neq kt\neq s$，因为(30)的两边都是 r,s,t 的多项式。我们的问题是计算

$$\sum_k\binom{r-kt}{k}\binom{s-(n-k)t}{n-k}\frac{r}{r-kt}\frac{s}{s-(n-k)t}$$

看起来，它比前边的可怕的问题还坏得多！然而，注意它同等式(26)有很强的相似性，也注意 $t=0$ 的情况。

我们试探着把

$$\binom{r-kt}{k}\frac{r}{r-kt}$$

改变成 $\binom{r-kt-1}{k-1}\frac{r}{k}$，只是，后者显得反而失去了同等式(26)的相似性。而且当 $k=0$ 时它出错。处理它的一个更好的方法是使用部分分式的技术，由此一个带有复杂分母的分式可以用带有较简单的分母的分式之和来代替。其实，我们有

$$\frac{1}{r-kt}\,\frac{1}{s-(n-k)t}=\frac{1}{r+s-nt}\left(\frac{1}{r-kt}+\frac{1}{s-(n-k)t}\right)$$

把这代入我们的求和式中，就得到

$$\frac{s}{r+s-nt}\sum_k\binom{r-kt}{k}\binom{s-(n-k)t}{n-k}\frac{r}{r-kt}+$$

$$\frac{r}{r+s-nt}\sum_k\binom{r-kt}{k}\binom{s-(n-k)t}{n-k}\frac{s}{s-(n-k)t}$$

因此如果在第二个式子中我们把 k 变成 $n-k$，等式(26)就可计算出这两个公式来；所希望的结果就立即得出了。恒等式(26)和(30)是由 H.A.Rothe 在*Formulæ de Serierum Reversione* (Leipzig: 1793)给出的。这些公式的特殊情况仍然屡屡被“发现”。关于这些恒等式的有趣历史及某些推广，请参看 H.W.Gould 和 J.Kaucký, *Journal of Combinatorial Theory* **1** (1966), 233～247。

例 5 确定 $a_0,a_1,a_2,\cdots$ 的值，使得对于所有非负整数 n，

$$n!=a_0+a_1n+a_2n(n-1)+a_3n(n-1)(n-2)+\cdots\tag{31}$$

解：在上一小节中不加证明地给出的等式 1.2.5-(11)提供了答案。暂且假装我们还不知道它。显然这个问题确实有一个解，因为我们可以置 $n=0$，且确定 a_0，再置 $n=1$ 并确定 a_1，等等。

首先我们借助二项式系数来写等式(31)：

$$n! = \sum_k \binom{n}{k} k!\, a_k \tag{32}$$

求解像这样对于 a_k 的隐式方程的问题叫做反演问题,我们将使用的技术也适用于类似的问题。

这个思想是以等式(23)的特殊情况 $s=0$ 为基础的:

$$\sum_k \binom{r}{k}\binom{k}{n}(-1)^{r-k} = \binom{0}{n-r} = \delta_{nr}, \qquad \text{整数 } n, \text{整数 } r \geqslant 0 \tag{33}$$

这个公式的重要性在于当 $n \neq r$ 时、这个和式为零;这使得能够求解我们的问题,因为如同在例 3 那样大量的项都消去了:

$$\sum_n n!\binom{m}{n}(-1)^{m-n} = \sum_n \sum_k \binom{n}{k} k!\, a_k \binom{m}{n}(-1)^{m-n} =$$

$$\sum_k k!\, a_k \sum_n \binom{n}{k}\binom{m}{n}(-1)^{m-n} =$$

$$\sum_k k!\, a_k \delta_{km} = m!\, a_m$$

注意我们是怎么才得到一个其中仅有一个值 a_m 出现的等式的——通过对于 $n=0,1,\cdots$,把等式(32)的适当倍数加在一起,现在我们有

$$a_m = \sum_{n\geqslant 0}(-1)^{m-n}\frac{n!}{m!}\binom{m}{n} = \sum_{0\leqslant n\leqslant m}\frac{(-1)^{m-n}}{(m-n)!} = \sum_{0\leqslant n\leqslant m}\frac{(-1)^n}{n!}$$

这就完成了对例 5 的求解。现在让我们更仔细地考察等式(33)的含义:当 r 和 m 为非负正数时,我们有

$$\sum_k \binom{r}{k}(-1)^{r-k}\left(c_0\binom{k}{0} + c_1\binom{k}{1} + \cdots + c_m\binom{k}{m}\right) = c_r$$

因为在求和之后,其它项都消失了。通过适当选择系数 c_i,我们可以把 k 的任何多项式表示成为带有上标 k 的二项式系数的一个和式。因此我们发现

$$\sum_k \binom{r}{k}(-1)^{r-k}(b_0 + b_1 k + \cdots + b_r k^r) = r!\, b_r, \qquad \text{整数 } r \geqslant 0 \tag{34}$$

其中 $b_0 + \cdots + b_r k^r$ 表示次数为 r 或更低次数的任意多项式。(数值分析专业的学生对此公式将不会太惊讶,因为 $\sum_k \binom{r}{k}(-1)^{r-k} f(x+k)$ 是函数 $f(x)$ 的"第 r 次差分"。)

使用等式(34),我们立即可得到许多其它的关系式,这些关系式乍看起来显得很复杂,而且通常都需要很长的证明,例如,

$$\sum_k \binom{r}{k}\binom{s-kt}{r}(-1)^k = t^r, \qquad \text{整数 } r \geq 0 \tag{35}$$

通常在像本书这样的教科书里，都给出大量灵巧变换的令人印象深刻的例子，但从不去提及这些技术失灵的看似简单的问题。以上给出的例子，可能给人们留下了通过二项式系数所有事情都办得到的印象。然而，应当指出，尽管有等式(10)，(11)和(18)，但对于类似的求和

$$\sum_{k=0}^{n}\binom{m}{k} = \binom{m}{0}+\binom{m}{1}+\cdots+\binom{m}{n} \tag{36}$$

当 $n < m$ 时，却没有简单的公式。（对于 $n = m$ 答案是简单的；它等于什么？见习题36。）

另一方面，当 m 是一个明显的负整数时，这个和式确实有作为 n 的函数的一个封闭形式，例如，

$$\sum_{k=0}^{n}\binom{-2}{k} = (-1)^n\left\lceil\frac{n+1}{2}\right\rceil \tag{37}$$

对于一个看似更难的，而不是更容易的一个和，也有一个简单的公式

$$\sum_{k=0}^{n}\binom{m}{k}\left(k-\frac{m}{2}\right) = -\frac{m}{2}\binom{m-1}{n} \tag{38}$$

我们如何判断什么时候该停止求解一个不能简化的和式呢？幸而，在许多重要情况下，都有一个好办法来回答这个问题。R.W.Gosper 和 D.Zeilberger 给出的一个算法，当二项式系数存在封闭形式时能发现它们，而当它们不存在时，能证明它们的不可能性。Gosper-Zeilberger 算法超出本书的范围，但在*CMath* §5.8 中作了说明。也可参看 Petkovšek，Wilf 及 Zeilberger 的书*A = B*（Wellesley，Mass.：A.K.Peters，1996）。

以系统的机械的方式处理二项式求和的重要工具是利用超几何函数的性质，超几何函数是借助于上升的阶乘幂如下定义的无穷级数：

$$F\left(\begin{matrix}a_1,\cdots,a_m\\ b_1,\cdots,b_n\end{matrix}\middle| z\right) = \sum_{k\geq 0}\frac{a_1^{\overline{k}}\cdots a_m^{\overline{k}}}{b_1^{\overline{k}}\cdots b_n^{\overline{k}}}\frac{z^k}{k!} \tag{39}$$

关于这些重要函数的介绍可在*CMath* 5.5 节和 5.6 节中找到。关于历史的文献，也可参见 J.Dutka，*Archive for History of Exact Sciences* **31**（1984），15 ~ 34。

二项式系数的概念有许多重要的推广，我们将简单地讨论它。首先，我们可以考虑在 $\binom{r}{k}$ 中的下标 k 的任意实数值；参见习题 40 至 45。我们还有推广

$$\binom{r}{k}_q = \frac{(1-q^r)(1-q^{r-1})\cdots(1-q^{r-k+1})}{(1-q^k)(1-q^{k-1})\cdots(1-q^1)} \tag{40}$$

当 q 趋近于极限值 1 时，它变成通常的二项式系数 $\binom{r}{k}_1 = \binom{r}{k}$；这可以通过以 $1-q$ 来除分子和分母中的每一项看出。这样的"q 项式系数"的基本性质在习题 58 中讨论。

然而，对于我们的目的说来，最重要的推广是多项式系数

$$\binom{k_1+k_2+\cdots+k_m}{k_1, k_2, \cdots, k_m} = \frac{(k_1+k_2+\cdots+k_m)!}{k_1!\,k_2!\cdots k_m!}, \quad \text{整数 } k_i \geqslant 0 \tag{41}$$

多项式系数的主要性质是等式(13)的推广：

$$(x_1+x_2+\cdots+x_m)^n = \sum_{k_1+k_2+\cdots+k_m=n} \binom{n}{k_1, k_2, \cdots, k_m} x_1^{k_1} x_2^{k_2} \cdots x_m^{k_m} \tag{42}$$

重要的是要注意到，多项式系数可以借助于二项式系数来表达：

$$\binom{k_1+k_2+\cdots+k_m}{k_1, k_2, \cdots, k_m} = \binom{k_1+k_2}{k_1}\binom{k_1+k_2+k_3}{k_1+k_2}\cdots\binom{k_1+k_2+\cdots+k_m}{k_1+k_2+\cdots+k_{m-1}} \tag{43}$$

所以我们可以应用对于处理二项式系数我们已知的技术。等式(20)两边都是三项式系数

$$\binom{r}{k, m-k, r-m}$$

我们以从 x 的幂表达的多项式到二项式系数表达的多项式的转换的简要分析，来结束本小节。在这一转换中涉及的系数叫做斯特林数。在许多算法的研究中都出现这些数。

斯特林数以两种风格出现：我们以 $\left[{n \atop k}\right]$ 来标记头一类斯特林数，而以 $\left\{{n \atop k}\right\}$ 来标记第二类斯特林数。这些记号是由 Jovan Karamata 给出的[*Mathematica*（Cluj）**9**（1935），164～178]，它们比起其它已被尝试过的符号具有令人信服的优点[参见 D. E. Knuth, *AMM* **99**（1992），403～422]。我们能记起在 $\left\{{n \atop k}\right\}$ 中的花括弧，因为花括弧表示集合，而 $\left\{{n \atop k}\right\}$ 是把 n 个元素的一个集合分划成为 k 个不相交的子集的方式数

(见习题 64)。其它的斯特林数$\left[{n \atop k}\right]$也有一个组合的解释,我们将在 1.3.3 小节中研究它:$\left[{n \atop k}\right]$是 n 个字母有 k 个循环的排列的个数。

表 2 列出了斯特林数,在某些方面它类似于帕斯卡三角。

表 2 两种类型的斯特林数

n	$\left[{n \atop 0}\right]$	$\left[{n \atop 1}\right]$	$\left[{n \atop 2}\right]$	$\left[{n \atop 3}\right]$	$\left[{n \atop 4}\right]$	$\left[{n \atop 5}\right]$	$\left[{n \atop 6}\right]$	$\left[{n \atop 7}\right]$	$\left[{n \atop 8}\right]$
0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0
3	0	2	3	1	0	0	0	0	0
4	0	6	11	6	1	0	0	0	0
5	0	24	50	35	10	1	0	0	0
6	0	120	274	225	85	15	1	0	0
7	0	720	1764	1624	735	175	21	1	0
8	0	5040	13068	13132	6769	1960	322	28	1
n	$\left\{{n \atop 0}\right\}$	$\left\{{n \atop 1}\right\}$	$\left\{{n \atop 2}\right\}$	$\left\{{n \atop 3}\right\}$	$\left\{{n \atop 4}\right\}$	$\left\{{n \atop 5}\right\}$	$\left\{{n \atop 6}\right\}$	$\left\{{n \atop 7}\right\}$	$\left\{{n \atop 8}\right\}$
0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0
3	0	1	3	1	0	0	0	0	0
4	0	1	7	6	1	0	0	0	0
5	0	1	15	25	10	1	0	0	0
6	0	1	31	90	65	15	1	0	0
7	0	1	63	301	350	140	21	1	0
8	0	1	127	966	1701	1050	266	28	1

关于当 n 很大时的有效近似值,参见 L.Moser 和 M.Wyman, *J.London Math.Soc.* **33** (1958),133 ~ 146; *Duke Math.J.* **25** (1958),29 ~ 43;D.E.Barton,F.N.David 及 M.Merrington,*Biometrika* **47** (1960),439 ~ 445;**50** (1963),169 ~ 176;N.M.Temme,*Studies in Applied Math.* **89** (1993),233 ~ 243;H.S.Wilf,*J.Combinatorial Theory* **A64** (1993),344 ~ 349;H.-K.Hwang(黄显贵),*J.Combinatorial Theory* **A71** (1995),343 ~ 351

第一类斯特林数用来把阶乘幂转换成通常的乘幂:

$$x^{\underline{n}} = x(x-1)\cdots(x-n+1) =$$

$$\left[{n \atop n}\right]x^n - \left[{n \atop n-1}\right]x^{n-1} + \cdots + (-1)^n\left[{n \atop 0}\right] =$$

$$\sum_k (-1)^{n-k} \genfrac{[}{]}{0pt}{}{n}{k} x^k \tag{44}$$

例如，由表 2，

$$\binom{x}{5} = \frac{x^{\underline{5}}}{5!} = \frac{1}{120}(x^5 - 10x^4 + 35x^3 - 50x^2 + 24x)$$

第二类斯特林数用来把通常的乘幂转换成阶乘幂：

$$x^n = \genfrac{\{}{\}}{0pt}{}{n}{n} x^{\underline{n}} + \cdots + \genfrac{\{}{\}}{0pt}{}{n}{1} x^{\underline{1}} + \genfrac{\{}{\}}{0pt}{}{n}{0} x^{\underline{0}} = \sum_k \genfrac{\{}{\}}{0pt}{}{n}{k} x^{\underline{k}} \tag{45}$$

这个公式事实上是斯特林在他的*Methodus Differentialis*（London：1730）中研究数$\genfrac{\{}{\}}{0pt}{}{n}{k}$的最初原因。例如，由表 2 我们有

$$x^5 = x^{\underline{5}} + 10x^{\underline{4}} + 25x^{\underline{3}} + 15x^{\underline{2}} + x^{\underline{1}} =$$

$$120\binom{x}{5} + 240\binom{x}{4} + 150\binom{x}{3} + 30\binom{x}{2} + \binom{x}{1}$$

我们现在将列出涉及斯特林数的最重要的恒等式。在这些等式中，变量 m 和 n 总是表示非负整数。

加法公式：

$$\genfrac{[}{]}{0pt}{}{n+1}{m} = n\genfrac{[}{]}{0pt}{}{n}{m} + \genfrac{[}{]}{0pt}{}{n}{m-1}$$

$$\genfrac{\{}{\}}{0pt}{}{n+1}{m} = m\genfrac{\{}{\}}{0pt}{}{n}{m} + \genfrac{\{}{\}}{0pt}{}{n}{m-1} \tag{46}$$

反演公式（同等式（33）相对照）：

$$\sum_k \genfrac{[}{]}{0pt}{}{n}{k}\genfrac{\{}{\}}{0pt}{}{k}{m}(-1)^{n-k} = \delta_{mn}, \quad \sum_k \genfrac{\{}{\}}{0pt}{}{n}{k}\genfrac{[}{]}{0pt}{}{k}{m}(-1)^{n-k} = \delta_{mn} \tag{47}$$

特殊值：

$$\binom{0}{n} = \genfrac{[}{]}{0pt}{}{0}{n} = \genfrac{\{}{\}}{0pt}{}{0}{n} = \delta_{n0}, \quad \binom{n}{n} = \genfrac{[}{]}{0pt}{}{n}{n} = \genfrac{\{}{\}}{0pt}{}{n}{n} = 1 \tag{48}$$

$$\genfrac{[}{]}{0pt}{}{n}{n-1} = \genfrac{\{}{\}}{0pt}{}{n}{n-1} = \binom{n}{2} \tag{49}$$

$$\genfrac{[}{]}{0pt}{}{n+1}{0} = \genfrac{\{}{\}}{0pt}{}{n+1}{0} = 0, \ \genfrac{[}{]}{0pt}{}{n+1}{1} = n!, \ \genfrac{\{}{\}}{0pt}{}{n+1}{1} = 1, \ \genfrac{\{}{\}}{0pt}{}{n+1}{2} = 2^n - 1 \tag{50}$$

展开公式：

$$\sum_k \begin{bmatrix} n \\ k \end{bmatrix} \binom{k}{m} = \begin{bmatrix} n+1 \\ m+1 \end{bmatrix}, \quad \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \binom{k}{m} (-1)^{k-m} = \begin{bmatrix} n \\ m \end{bmatrix} \tag{51}$$

$$\sum_k \begin{Bmatrix} k \\ m \end{Bmatrix} \binom{n}{k} = \begin{Bmatrix} n+1 \\ m+1 \end{Bmatrix}, \quad \sum_k \begin{Bmatrix} k+1 \\ m+1 \end{Bmatrix} \binom{n}{k} (-1)^{n-k} = \begin{Bmatrix} n \\ m \end{Bmatrix} \tag{52}$$

$$\sum_k \binom{m}{k} (-1)^{m-k} k^n = m! \begin{Bmatrix} n \\ m \end{Bmatrix} \tag{53}$$

$$\begin{aligned} \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{Bmatrix} m+k \\ k \end{Bmatrix} &= \begin{bmatrix} n \\ n-m \end{bmatrix} \\ \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{bmatrix} m+k \\ k \end{bmatrix} &= \begin{Bmatrix} n \\ n-m \end{Bmatrix} \end{aligned} \tag{54}$$

$$\sum_k \begin{Bmatrix} n+1 \\ k+1 \end{Bmatrix} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^{k-m} = \binom{n}{m} \tag{55}$$

$$\sum_{k \leqslant n} \begin{bmatrix} k \\ m \end{bmatrix} \frac{n!}{k!} = \begin{bmatrix} n+1 \\ m+1 \end{bmatrix}, \quad \sum_{k \leqslant n} \begin{Bmatrix} k \\ m \end{Bmatrix} (m+1)^{n-k} = \begin{Bmatrix} n+1 \\ m+1 \end{Bmatrix} \tag{56}$$

其它一些基本的斯特林恒等式出现在习题 1.2.6-61 和 1.2.7-6，以及 1.2.9 小节的等式(23)，(26)，(27)和(28)中。

等式(49)刚好是一般现象的一种情况：当 m 是非负整数时，斯特林数 $\begin{bmatrix} n \\ n-m \end{bmatrix}$ 和 $\begin{Bmatrix} n \\ n-m \end{Bmatrix}$ 是 $2m$ 次的 n 的多项式。例如，对于 $m=2$ 和 $m=3$ 的公式是

$$\begin{bmatrix} n \\ n-2 \end{bmatrix} = \binom{n}{4} + 2\binom{n+1}{4}; \quad \begin{Bmatrix} n \\ n-2 \end{Bmatrix} = \binom{n+1}{4} + 2\binom{n}{4}$$

$$\begin{bmatrix} n \\ n-3 \end{bmatrix} = \binom{n}{6} + 8\binom{n+1}{6} + 6\binom{n+2}{6}; \quad \begin{Bmatrix} n \\ n-3 \end{Bmatrix} = \binom{n+2}{6} + 8\binom{n+1}{6} + 6\binom{n}{6} \tag{57}$$

因此对于 r 的任意的实数(或复数)值，定义数 $\begin{bmatrix} r \\ r-m \end{bmatrix}$ 和 $\begin{Bmatrix} r \\ r-m \end{Bmatrix}$ 是有意义的。通过这个推广，以一个有趣的对偶律把两种斯特林数联系起来：

$$\begin{Bmatrix} n \\ m \end{Bmatrix} = \begin{bmatrix} -m \\ -n \end{bmatrix} \tag{58}$$

它蕴涵于斯特林最初的讨论中。而且，等式(45)在 z 的实部为正时无穷级数

$$z^r = \sum_k \left\{ {r \atop r-k} \right\} z^{\underline{r-k}} \tag{59}$$

收敛的意义下，一般都保持成立。相伴随的公式，即等式(44)，以类似方式推广到一个渐近的(但不收敛)的序列(参见习题 65)：

$$z^{\underline{r}} = \sum_{k=0}^{m} \left[{r \atop r-k} \right] (-1)^k z^{r-k} + O(z^{r-m-1}) \tag{60}$$

CMath 的 6.1,6.2 和 6.5 节包含有关于斯特林数以及在公式中如何处理它们的附加的信息。关于包括斯特林数作为特例的一般的三角形族，也请参见习题 4.7-21。

习　题

1. [*00*] 从 n 个事物中每次取 $n-1$ 个有多少种可能的组合？

2. [*00*] $\binom{0}{0}$ 等于什么？

3. [*00*] 可能有多少种桥牌牌型(即从 52 张牌取 13 张牌)？

4. [*10*] 把习题 3 的答案以素数的乘积给出。

▶ 5. [*05*] 使用帕斯卡三角说明 $11^4 = 14641$ 这一事实。

▶ 6. [*10*] 通过使用加法公式(9)，帕斯卡三角(表 1)可以沿所有方向推广。试求出表 1 顶部往上去三行(即是，对于 $r=-1,-2$ 和 -3 的情形)。

7. [*12*] 如果 n 是固定的正整数，k 的什么值使 $\binom{n}{k}$ 成为极大值？

8. [*00*] 在"对称条件"等式(6)中，反映了帕斯卡三角的什么性质？

9. [*01*] $\binom{n}{n}$ 的值是多少？(考虑所有整数 n。)

▶ 10. [*M25*] 如果 p 是素数，证明

a) $\binom{n}{p} \equiv \left\lfloor \frac{n}{p} \right\rfloor$ (modulo p)。

b) $\binom{p}{k} \equiv 0$ (modulo p)，对于 $1 \le k \le p-1$。

c) $\binom{p-1}{k} \equiv (-1)^k$　(modulo p)，对于 $0 \le k \le p-1$。

d) $\binom{p+1}{k} \equiv 0$ (modulo p)，对于 $2 \le k \le p-1$。

e) (E. Lucas, 1877)

$$\binom{n}{k} \equiv \binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor} \binom{n \bmod p}{k \bmod p} \quad (\text{modulo} \quad p)$$

f) 如果 n 和 k 的 p 进制表示为

$$n = a_r p^r + \cdots + a_1 p + a_0$$

$$k = b_r p^r + \cdots + b_1 p + b_0$$

则 $\binom{n}{k} \equiv \binom{a_r}{b_r} \cdots \binom{a_1}{b_1} \binom{a_0}{b_0} \pmod{p}$。

▶ 11. [*M20*] (E. Kummer, 1852) 设 p 为素数。证明若 p^n 能整除

$$\binom{a+b}{a}$$

但 p^{n+1} 不能整除，则 n 就等于在 p 进制系统中，a 加 b 所出现的进位个数[提示：参见习题 1.2.5-12。]

12. [*M22*] 有无这样的正整数 n 存在，它使得帕斯卡三角的第 n 行中所有非零项都是奇数。若有，试把所有这样的 n 找出来。

13. [*M13*] 证明求和公式(10)。

14. [*M21*] 计算 $\sum_{k=0}^{n} k^4$。

15. [*M15*] 证明二项式公式等式(13)。

16. [*M15*] 给定 n 和 k 为正整数，证明对称的恒等式

$$(-1)^n \binom{-n}{k-1} = (-1)^k \binom{-k}{n-1}$$

▶ 17. [*M18*] 利用 $(1+x)^{r+s} = (1+x)^r(1+x)^s$ 的思想，由等式(15)证明 Chu-Vandermonde 公式(21)。

18. [*M15*] 利用等式(21)和(6)，证明等式(22)。

19. [*M18*] 用归纳法证明等式(23)。

20. [*M20*] 利用等式(21)和(19)证明等式(24)，然后证明再使用等式(19)，可得等式(25)。

▶ 21. [*M05*] 等式(25)的两边都是 s 的多项式；为什么该等式不是关于 s 的恒等式？

22. [*M20*] 对于 $s = n - 1 - r + nt$ 的特殊情况，来证明等式(26)。

23. [*M13*] 假定等式(26)对于 (r, s, t, n) 和 $(r, s-t, t, n-1)$ 成立，证明对于 $(r, s+1, t, n)$ 它也成立。

24. [*M15*] 说明为什么把上两题的结果结合起来，就给出等式(26)的一个证明。

25. [*HM30*] 设多项式 $A_n(x,t)$ 之定义如等式(30)所示。设 $z = x^{t+1} - x^t$。证明，假定 z 足够小，$\sum_k A_k(r,t) z^k = x^r$。[注：如果 $t = 0$，这个结果实际上就是二项式定理，而且这个等式是二项式定理的一个重要的推广。在证明中可假定二项式定理(15)成立。]

提示：由恒等式

$$\sum_j (-1)^j \binom{k}{j} \binom{r-jt}{k} \frac{r}{r-jt} = \delta_{k0}$$

开始。

26. [*HM25*] 利用上题的假设，证明

$$\sum_k \binom{r-kt}{k} z^k = \frac{x^{r+1}}{(t+1)x - t}$$

27. [*HM21*] 利用习题 25 的结果解正文中的例 4;并由上两题证明等式(26)。[提示:参看习题 17。]

28. [*M25*] 如果 n 是非负整数,证明

$$\sum_k \binom{r+tk}{k}\binom{s-tk}{n-k} = \sum_{k\geq 0}\binom{r+s-k}{n-k}t^k$$

29. [*M20*] 说明等式(34)仅仅是习题 1.2.3-33 中证明的一般恒等式的一个特殊情况。

▶ 30. [*M24*] 通过把求和处理成可以应用等式(26)的形式,说明有比在正文中所使用的方法更好的方法来求解例 3。

▶ 31. [*M20*] 给定 m 和 n 都是整数,借助于 r,s,m 和 n 计算

$$\sum_k \binom{m-r+s}{k}\binom{n+r-s}{n-k}\binom{r+k}{m+n}$$

通过以

$$\sum_j \binom{r}{m+n-j}\binom{k}{j}$$

替换 $\binom{r+k}{m+n}$ 开始。

32. [*M20*] 证明 $\sum_k \left[{n \atop k}\right] x^k = x^{\overline{n}}$,其中 $x^{\overline{n}}$ 是在等式 1.2.5-(19)中定义的上升阶乘幂。

33. [*M20*] (Capelli 和数) 证明,当涉及的是上升的阶乘幂而不是通常的乘幂时,二项式公式也还成立。即证明恒等式 $(x+y)^{\overline{n}} = \sum_k \binom{n}{k} x^{\overline{k}} y^{\overline{n-k}}$

34. [*M23*](Torelli 和数) 按照上题的方法,证明对于上升的阶乘幂,阿贝尔对于二项式公式的推广,即等式(16)也成立:

$$(x+y)^{\overline{n}} = \sum \binom{n}{k} x(x-kz+1)^{\overline{k-1}}(y+kz)^{\overline{n-k}}$$

35. [*M23*] 直接由定义,等式(44)和(45),证明斯特林数的加法公式(46)。

36. [*M10*] 在帕斯卡三角的每行中数的和 $\sum_k \binom{n}{k}$ 等于什么? 带有交替符号的这些数之和 $\sum \binom{n}{k}(-1)^k$ 等于什么?

37. [*M10*] 从上题的答案,推导出在一行中每隔一项之和的值,即 $\binom{n}{0}+\binom{n}{2}+\binom{n}{4}+\cdots$。

38. [*HM30*](C. Ramus, 1834) 推广上题的结果,证明给定 $0\leq k<m$,我们有公式

$$\binom{n}{k}+\binom{n}{m+k}+\binom{n}{2m+k}+\cdots = \frac{1}{m}\sum_{0\leq j<m}\left(2\cos\frac{j\pi}{m}\right)^n \cos\frac{j(n-2k)\pi}{m}$$

例如,

$$\binom{n}{1}+\binom{n}{4}+\binom{n}{7}+\cdots = \frac{1}{3}\left(2^n+2\cos\frac{(n-2)\pi}{3}\right)$$

[提示:求出这些系数乘以 m 次单位根的正确组合。]当 $m \geqslant n$ 时,这个恒等式是特别值得注意的。

39. [*M10*] 在第一种斯特林三角的每行中诸数之和 $\sum_k \left[{n \atop k}\right]$ 等于什么?带有交替的正负号的这些数之和等于什么?(参见习题 36。)

40. [*HM17*] 对于正实数 x,y,贝塔函数 $\mathrm{B}(x,y)$ 由公式 $\mathrm{B}(x,y)=\int_0^1 t^{x-1}(1-t)^{y-1}\mathrm{d}t$ 定义。

a) 证明 $\mathrm{B}(x,1)=\mathrm{B}(1,x)=1/x$。

b) 证明 $\mathrm{B}(x+1,y)+\mathrm{B}(x,y+1)=\mathrm{B}(x,y)$。

c) 证明 $\mathrm{B}(x,y)=((x+y)/y)\mathrm{B}(x,y+1)$。

41. [*HM22*] 如果 m 是正整数,通过证明 $\Gamma_m(x)=m^x\mathrm{B}(x,m+1)$,我们证明了伽玛函数与习题 1.2.5-19 中的贝塔函数之间的一个关系。

a) 证明

$$\mathrm{B}(x,y)=\frac{\Gamma_m(y)m^x}{\Gamma_m(x+y)}\mathrm{B}(x,y+m+1)$$

b) 证明

$$\mathrm{B}(x,y)=\frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$$

42. [*HM10*] 借助于上边定义的贝塔函数表达二项式系数 $\binom{r}{k}$。(这就给出了把二项式系数的定义推广到所有实数值 k 的办法。)

43. [*HM20*] 证明 $\mathrm{B}(1/2,1/2)=\pi$。(由习题 41,我们现在可以作出结论 $\Gamma(1/2)=\sqrt{\pi}$。)

44. [*HM20*] 使用习题 42 中所提出的推广的二项式系数,证明

$$\binom{r}{1/2}=2^{2r+1}\bigg/\binom{2r}{r}\pi$$

45. [*HM21*] 利用习题 42 中所提出的推广的二项式系数,求 $\lim_{r\to\infty}\binom{r}{k}\Big/r^k$。

▶ 46. [*M21*] 利用斯特林近似公式等式 1.2.5-(7),试求 $\binom{x+y}{y}$ 的近似值,假定 x 和 y 都充分大。特别地,试求当 n 充分大时,$\binom{2n}{n}$ 的近似数值。

47. [*M21*] 给定 k 为一整数,证明

$$\binom{r}{k}\binom{r-1/2}{k}=\binom{2r}{k}\binom{2r-k}{k}\bigg/4^k=\binom{2r}{2k}\binom{2k}{k}\bigg/4^k$$

对于 $r=-1/2$ 的特殊情况给出一个更简单的公式。

▶ 48. [*M25*] 如果分母非零,证明

$$\sum_{k\geq 0}\binom{n}{k}\frac{(-1)^k}{k+x}=\frac{n!}{x(x+1)\cdots(x+n)}=\frac{1}{x\binom{n+x}{n}}$$

[注意:这个公式给出了二项式系数的倒数,以及 $1/x(x+1)\cdots(x+n)$的部分分式展开。]

49. [*M20*] 证明恒等式$(1+x)^r=(1-x^2)^r(1-x)^{-r}$隐含了二项式系数的一个关系。

50. [*M20*] 在 $x+y=0$ 的特殊情况下,证明阿贝尔公式(16)。

51. [*M21*] 通过写 $y=(x+y)-x$,把右边展开成$(x+y)$的乘幂,并应用上题的结果,证明阿贝尔公式(16)。

52. [*HM11*] 当 n 不是非负整数时,通过当 $n=x=-1,y=z=1$ 时计算右边的值,证明阿贝尔二项式公式(16)并不总是成立的。

53. [*M25*] (a) 对 m 用归纳法,证明下列恒等式

$$\sum_{k=0}^{m}\binom{r}{k}\binom{s}{n-k}(nr-(r+s)k)=(m+1)(n-m)\binom{r}{m+1}\binom{s}{n-m}$$

其中 m 和 n 是整数。

(b) 利用来自习题 47 的重要关系

$$\binom{-1/2}{n}=\frac{(-1)^n}{2^{2n}}\binom{2n}{n},\quad \binom{1/2}{n}=\frac{(-1)^{n-1}}{2^{2n}(2n-1)}\binom{2n}{n}=\frac{(-1)^{n-1}}{2^{2n-1}(2n-1)}\binom{2n-1}{n}-\delta_{n0}$$

证明下列公式可作为(a)中的恒等式的特殊情况得到:

$$\sum_{k=0}^{m}\binom{2k-1}{k}\binom{2n-2k}{n-k}\frac{-1}{2k-1}=\frac{n-m}{2n}\binom{2m}{m}\binom{2n-2m}{n-m}+\frac{1}{2}\binom{2n}{n}$$

(这一结果比起在 $r=-1,s=0,t=-2$ 情况下的等式(26)要更为一般得多。)

54. [*M21*] 把帕斯卡三角(如表 1 所示)当做一个矩阵,此矩阵的逆是什么?

55. [*M21*] 把斯特林三角(表 2)的每一个都当做矩阵,试确定它们的逆。

56. [*20*](组合数系统) 对于每一个整数 $n=0,1,2,\cdots,20$,试求三个整数 a,b,c,使得 $n=\binom{a}{3}+\binom{b}{2}+\binom{c}{1}$,且 $a>b>c\geq 0$。你能否看出,对于更高的 n 值,如何继续下去?

▶ 57. [*M22*] 说明斯特林在推广阶乘函数等式 1.2.5-(12)的尝试中的系数 a_m 是

$$\frac{(-1)^m}{m!}\sum_{k\geq 1}(-1)^k\binom{m-1}{k-1}\ln k$$

58. [*M23*] 在等式(40)的记号下,证明"q 项式定理"

$$(1+x)(1+qx)\cdots(1+q^{n-1}x)=\sum_k\binom{n}{k}_q q^{k(k-1)/2}x^k$$

试求基本恒等式(17)和(21)的 q 项式推广。

59. [*M25*] 一数列 A_{nk}, $n\geq 0$, $k\geq 0$,对于 $nk>0$ 满足关系 $A_{n0}=1$, $A_{0k}=\delta_{0k}$, $A_{nk}=A_{(n-1)k}+A_{(n-1)(k-1)}+\binom{n}{k}$。试求 A_{nk}。

▶ 60. [*M23*] 我们已经看到，$\binom{n}{k}$是从 n 个事物中每次取 k 个的组合的数目，即从 n 个事物的集合中选出 k 个不同的事物的方法种数。有重复的组合类似于通常的组合，所不同的只是我们可以任意多次地选择每一个事物，因此，如果我们考虑有重复的组合的话，列表(1)将被扩充成包括 *aaa*，*aab*，*aac*，*aad*，*aae*，*abb* 等等。如果允许重复，问从 n 个事物中取出 k 个的组合，共有多少种?

▶ 61. [*M25*] 计算和数

$$\sum_k \left[{n+1 \atop k+1} \right] \left\{ {k \atop m} \right\} (-1)^{k-m}$$

由此得到一个与等式(55)配对的公式。

▶ 62. [*M23*] 正文中给出了涉及两个二项式系数的乘积的和数的公式，在涉及三个二项式系数的乘积的和数中，下列公式和习题 31 的恒等式似乎是最有用的：

$$\sum_k (-1)^k \binom{l+m}{l+k} \binom{m+n}{m+k} \binom{n+l}{n+k} = \frac{(l+m+n)!}{l!\,m!\,n!}, \qquad \text{整数 } l,m,n \geqslant 0$$

(这个和式包括了正的和负的 k 值。)试证明此恒等式。[提示：从应用习题 31 的结果开始，有一个非常短的证明。]

63. [*M30*] 如果 l，m 和 n 都是整数且 $n \geqslant 0$，证明

$$\sum_{j,k} (-1)^{j+k} \binom{j+k}{k+l} \binom{r}{j} \binom{n}{k} \binom{s+n-j-k}{m-j} = (-1)^l \binom{n+r}{n+l} \binom{s-r}{m-n-l}$$

▶ 64. [*M20*] 证明$\left\{ {n \atop m} \right\}$是把 n 个元素的一个集合分划成为 m 个非空的不相交子集的方式数。例如集合{1,2,3,4}可以以$\left\{ {4 \atop 2} \right\}=7$ 种方式分划成两个元素的子集：{1,2,3}{4}，{1,2,4}{3}，{1,3,4}{2}，{2,3,4}{1}，{1,2}{3,4}，{1,3}{2,4}，{1,4}{2,3}。提示：使用等式(46)。

65. [*HM35*](B.F.Logan) 证明等式(59)和(60)。

66. [*HM30*] 设 x，y 和 z 是满足

$$\binom{x}{n} = \binom{y}{n} + \binom{z}{n-1}$$

的实数，其中 $x \geqslant n-1$，$y \geqslant n-1$，$z \geqslant n-2$，且 $n \geqslant 2$ 是整数。证明

$$\binom{x}{n-1} \leqslant \binom{y}{n-1} + \binom{z}{n-2} \qquad \text{当且仅当} \qquad y \geqslant z$$

$$\binom{x}{n+1} \leqslant \binom{y}{n+1} + \binom{z}{n} \qquad \text{当且仅当} \qquad y \leqslant z$$

▶67. [*M20*] 我们通常需要知道二项式系数都不是太大的。试证明易于记忆的上限，当 $n \geqslant$

$k \geqslant 0$时$\binom{n}{k} \leqslant \left(\frac{ne}{k}\right)^k$。

68. [*M25*] (A. De Moivre) 试证明，如果 n 是非负整数，则

$$\sum_k \binom{n}{k} p^k (1-p)^{n-k} |k - np| = 2\lceil np \rceil \binom{n}{\lceil np \rceil} p^{\lceil np \rceil} (1-p)^{n+1-\lceil np \rceil}$$

1.2.7　调和数

在我们后面的工作中，下列和数将具有巨大的重要性：

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k}, \quad n \geqslant 0 \tag{1}$$

在经典数学中，这个和数并不经常出现，因此对它没有标准的记号；但在算法分析中，动不动它就要冒出来，因此我们将自始至终地把它叫做 H_n。(除了 H_n 之外，在数学著作中还可找到 h_n，s_n，以及 $\psi(n+1)+\gamma$ 的记号。字母 H 代表"调和的"，我们称 H_n 为调和数，因为(1)习惯上称调和级数。)

乍看起来，当 n 有一个很大的值时，H_n 未必会成为太大的值，因为我们总是加上越来越小的数。但是，实际上不难看出，如果我们把 n 取得足够大，H_n 就可能变成我们高兴它多大它就有多大，因为

$$H_{2^m} \geqslant 1 + \frac{m}{2} \tag{2}$$

这一下限是从下列观察得出的，对于 $m \geqslant 0$，我们有

$$H_{2^{m+1}} = H_{2^m} + \frac{1}{2^m+1} + \frac{1}{2^m+2} + \cdots + \frac{1}{2^{m+1}} \geqslant$$

$$H_{2^m} + \frac{1}{2^{m+1}} + \frac{1}{2^{m+1}} + \cdots + \frac{1}{2^{m+1}} = H_{2^m} + \frac{1}{2}$$

所以随着 m 加 1，(2)的左边至少增加$\frac{1}{2}$。

重要的是对于 H_n 的值，有比在等式(2)中所给出的更为详细的信息。H_n 的近似大小是一个熟知的数量(至少在数学界是这样)，它可以表达如下：

$$H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \epsilon, \quad 0 < \epsilon < \frac{1}{252n^6} \tag{3}$$

这里 $\gamma = 0.5772156649\cdots$ 是欧拉常数，是由欧拉在*Commentarii Acad. Sci. Imp. Pet.* **7** (1734)，150 ~ 161 上引入的。附录 A 的表格中给出了对于小的 n 的 H_n 的值，以及 γ 的一个 40 位的值。我们将在 1.2.11.2 小节中推导等式(3)。

因此，H_n 相当接近于 n 的自然对数。习题 7(a)以简单的方式揭示 H_n 略具对数的性质。

在某种意义下，当 n 取充分大值时，H_n 仅勉强地趋于无穷大，因为当 r 是任何大于 1 的实数值时，

$$1+\frac{1}{2^r}+\frac{1}{3^r}+\cdots+\frac{1}{n^r} \tag{4}$$

对于所有的 n 都保持有界(参见习题 3)。我们以 $H_n^{(r)}$ 来标记等式(4)中的和数。

当等式(4)中的指数 r 至少为 2 时，$H_n^{(r)}$ 的值相当接近于它的极大值 $H_\infty^{(r)}$，除非 n 的值很小。量 $H_\infty^{(r)}$ 在数学上作为黎曼(Riemann)ζ塔(zeta)函数而非常有名：

$$H_\infty^{(r)}=\zeta(r)=\sum_{k\geq 1}\frac{1}{k^r} \tag{5}$$

如果 r 是一个偶整数，已知 $\zeta(r)$ 等于

$$H_\infty^{(r)}=\frac{1}{2}\,|B_r|\,\frac{(2\pi)^r}{r!},\qquad \text{整数 } r/2\geq 1 \tag{6}$$

其中 B_r 是一个伯努利数(参见 1.2.11.2 小节和附录 A)。特别是

$$H_\infty^{(2)}=\frac{\pi^2}{6},\ H_\infty^{(4)}=\frac{\pi^4}{90},\ H_\infty^{(6)}=\frac{\pi^6}{945},\ H_\infty^{(8)}=\frac{\pi^8}{9450} \tag{7}$$

这些值都是由欧拉给出的；有关的讨论和证明参见*CMath*，§ 6.5。

现在我们将考虑一些涉及调和数的重要和数。首先，

$$\sum_{k=1}^{n}H_k=(n+1)H_n-n \tag{8}$$

这可从求和的简单交换得出：

$$\sum_{k=1}^{n}\sum_{j=1}^{k}\frac{1}{j}=\sum_{j=1}^{n}\sum_{k=j}^{n}\frac{1}{j}=\sum_{j=1}^{n}\frac{n+1-j}{j}$$

公式(8)是和数 $\sum_{k=1}^{n}\binom{k}{m}H_k$ 的一个特殊情况，我们现在利用称做分部求和的重要技术(参见习题 10)来确定这一和数。每当 $\sum a_k$ 和 $(b_{k+1}-b_k)$ 都有简单形式时，分部求和是计算 $\sum a_kb_k$ 的一个有用方法。在此情况下，我们发现

$$\binom{k}{m}=\binom{k+1}{m+1}-\binom{k}{m+1}$$

且因此

$$\binom{k}{m}H_k=\binom{k+1}{m+1}\left(H_{k+1}-\frac{1}{k+1}\right)-\binom{k}{m+1}H_k$$

因此

$$\sum_{k=1}^{n}\binom{k}{m}H_k=\left(\binom{2}{m+1}H_2-\binom{1}{m+1}H_1\right)+\cdots+$$

$$\left(\binom{n+1}{m+1}H_{n+1}-\binom{n}{m+1}H_n\right)-\sum_{k=1}^{n}\binom{k+1}{m+1}\frac{1}{k+1}=$$

$$\binom{n+1}{m+1}H_{n+1}-\binom{1}{m+1}H_1-\frac{1}{m+1}\sum_{k=0}^{n}\binom{k}{m}+\frac{1}{m+1}\binom{0}{m}$$

应用等式 1.2.6-(11)得出所需要的公式

$$\sum_{k=1}^{n}\binom{k}{m}H_k=\binom{n+1}{m+1}\left(H_{n+1}-\frac{1}{m+1}\right) \tag{9}$$

(这一推导和它最后的结果类似于使用微积分学书籍称为分部积分的技术对

$$\int_1^n x^m \ln x\, dx=\frac{n^{m+1}}{m+1}\left(\ln\ n-\frac{1}{m+1}\right)+\frac{1}{(m+1)^2}$$

的计算。)

我们通过考虑不同类型的和数$\sum_k\binom{n}{k}x^kH_k$来结束本小节。为简便起见,暂时以 S_n 表示这个和数,我们发现

$$S_{n+1}=\sum_k\left(\binom{n}{k}+\binom{n}{k-1}\right)x^kH_k=S_n+x\sum_{k\geq 1}\binom{n}{k-1}x^{k-1}\left(H_{k-1}+\frac{1}{k}\right)=$$

$$S_n+xS_n+\frac{1}{n+1}\sum_{k\geq 1}\binom{n+1}{k}x^k$$

因此 $S_{n+1}=(x+1)S_n+((x+1)^{n+1}-1)/(n+1)$,我们有

$$\frac{S_{n+1}}{(x+1)^{n+1}}=\frac{S_n}{(x+1)^n}+\frac{1}{n+1}-\frac{1}{(n+1)(x+1)^{n+1}}$$

这个等式连同 $S_1=x$ 的事实一起,向我们表明

$$\frac{S_n}{(x+1)^n}=H_n-\sum_{k=1}^{n}\frac{1}{k(x+1)^k} \tag{10}$$

剩下的和数是 $\ln(1/(1-1/(x+1)))=\ln(1+1/x)$的无穷级数 1.2.9-(17)的一部分,而且当 $x>0$ 时这个级数收敛;差是

$$\sum_{k>n}\frac{1}{k(x+1)^k}<\frac{1}{(n+1)(x+1)^{n+1}}\sum_{k\geq 0}\frac{1}{(x+1)^k}=\frac{1}{(n+1)(x+1)^n x}$$

这就证明了下列定理:

定理 A *如果 $x>0$,则*

$$\sum_{k=1}^{n}\binom{n}{k}x^kH_k=(x+1)^n\left(H_n-\ln\left(1+\frac{1}{x}\right)\right)+\epsilon$$

其中$0<\epsilon<1/(x(n+1))$ ▌

习 题

1. [*01*] H_0,H_1 和 H_2 等于什么?

2. [*13*] 说明正文中用来证明 $H_{2^m}\geqslant 1+m/2$ 的简单论证,只要稍加修改,即可证明 $H_{2^m}\leqslant 1+m$。

3. [*M21*] 推广上题所使用的论证来证明,假定 $r>1$,则 $H_n^{(r)}$ 对于所有的 n 保持有界。并求出一个上界来。

▶ 4. [*10*] 对于所有正整数 n,判定下列命题中哪个为真:(a) $H_n<\ln n$;(b) $H_n>\ln n$;(c) $H_n>\ln n+\gamma$。

5. [*15*] 利用附录 A 中的表,给出 H_{10000}的值到 15 位小数。

6. [*M15*] 证明调和数直接同上一小节所介绍的斯特林数有关;事实上

$$H_n=\left[{n+1 \atop 2}\right]\Big/ n!$$

7. [*M21*] 设 $T(m,n)=H_m+H_n-H_{mn}$。(a) 证明当 m 或 n 增加时,$T(m,n)$并不增加(假设 m 和 n 皆为正)。(b)对于 $m,n>0$,计算 $T(m,n)$的极小值和极大值。

8. [*HM18*] 将等式(8)与$\sum_{k=1}^{n}\ln k$ 进行比较;作为 n 的函数估计它们的差。

▶ 9. [*M18*] 仅当 $x>0$ 时,定理 A 才可成立,当 $x=-1$ 时所考虑的和数之值等于什么?

10. [*M20*](分部求和) 在习题 1.2.4-42 和等式(9)的推导中我们已经使用了分部求和的一般方法的特殊情况。试证一般的公式

$$\sum_{1\leqslant k<n}(a_{k+1}-a_k)b_k=a_nb_n-a_1b_1-\sum_{1\leqslant k<n}a_{k+1}(b_{k+1}-b_k)$$

▶ 11. [*M21*] 利用分部求和,计算 $\sum_{1<k\leqslant n}\frac{1}{k(k-1)}H_k$。

▶ 12. [*M10*] 试计算 $H_\infty^{(1000)}$精确到至少 100 位小数。

13. [*M22*] 证明恒等式

$$\sum_{k=1}^{n}\frac{x^k}{k}=H_n+\sum_{k=1}^{n}\binom{n}{k}\frac{(x-1)^k}{k}$$

(特别注意 $x=0$ 的特殊情况,它向我们提供了一个与习题 1.2.6-48 有关的恒等式。)

14. [*M22*] 证明$\sum_{k=1}^{n}H_k/k=\frac{1}{2}(H_n^2+H_n^{(2)})$,并计算$\sum_{k=1}^{n}H_k/(k+1)$。

▶ 15. [*M23*] 借助于 n 和 H_n 来表达$\sum_{k=1}^{n}H_k^2$。

16. [*18*] 借助于调和数来表达和数 $1+\frac{1}{3}+\cdots+\frac{1}{2n-1}$。

17. [*M24*](E. Waring, 1782) 设 p 是奇素数,试证明 H_{p-1}的分子可为 p 所整除。

18. [*M33*](J. Selfridge) 能整除 $1+\frac{1}{3}+\cdots+\frac{1}{2n-1}$的分子的 2 的最高次幂是多少?

▶ 19. [*M30*] 列出使 H_n 为一整数的所有非负整数 n。[提示:如果 H_n 有奇分子和偶分母,则

它不可能是一个整数。]

20. [*HM22*] 有一个和在本小节中导出定理 A 相同的解决求和问题的解析方法:如果 $f(x)=\sum_{k\geq 0}a_kx^k$,而且此级数对于 $x=x_0$ 收敛,试证

$$\sum_{k\geq 0}a_kx_0^kH_k=\int_0^1\frac{f(x_0)-f(x_0y)}{1-y}dy$$

21. [*M24*] 试计算 $\sum_{k=1}^{n}H_k/(n+1-k)$。

22. [*M28*] 试计算 $\sum_{k=0}^{n}H_kH_{n-k}$。

▶ 23. [*HM20*] 通过考虑函数 $\Gamma'(x)/\Gamma(x)$,说明我们怎样才能得到 H_n 对于 n 的非整数值的一个自然的推广。你可预先利用下道习题,并使用 $\Gamma'(1)=-\gamma$ 这一事实。

24. [*HM21*] 证明 $xe^{\gamma x}\prod_{k\geq 1}\left(\left(1+\frac{x}{k}\right)e^{-x/k}\right)=\frac{1}{\Gamma(x)}$。

(考虑这个无穷乘积的部分乘积。)

1.2.8 斐波那契数

序列

$$0,1,1,2,3,5,8,13,21,34\cdots \tag{1}$$

中,每个数都是它前面两项之和。这个序列在我们后面将要研究的许多表面上似乎无关的算法中,起着重要的作用。兹以 F_n 来表示这个序列的数,并将其形式地定义为:

$$F_0=0;\ F_1=1;\ F_{n+2}=F_{n+1}+F_n,\qquad n\geq 0 \tag{2}$$

这个著名的序列是 1202 年由伦纳德·皮萨诺(Leonardo Pisano,意为比萨的伦纳德)发表的。他有时又叫做伦纳德·斐波那契(Leonardo Fibonacci, Filius Bonaccii, Bonaccio 的儿子之意)。他所著的*Liber Abaci* (珠算书)包含下列习题:“在一年的时间里,一对兔子可以繁衍出多少对兔子?”为解决这个问题,我们被告知:假定每对兔子每个月生育一对兔子,新的一对兔子在一个月之后具有生育能力;其次,这些兔子都不死去。一个月后,将有 2 对兔子;两个月后,将有 3 对;下一个月原来的一对以及在头一个月生下的一对都将生出一对来,因而总共将有 5 对;等等。

斐波那契是中世纪以来欧洲最伟大的数学家。他研究了 al-Khwārizmī(在此人之后,就有了“算法”(algorithm)一词,见 1.1 节)的工作,而且对算术和几何都做出了许多开创性的贡献。斐波那契的著作 1857 年曾再版[B. Boncompagni,*Scritti di Leonardo Pisano* (Rome,1857—1862),两卷;F_n 出现在第 1 卷 283 ~ 285 页上]。当然,他的兔子问题并不是对生物学和人口增长问题的实际应用,它只是加法中的一个习题。事实上,它还是关于加法的一个比较好的计算机习题(参见习题 3);斐波那契写到:“这种加法可以按照这一顺序无穷多个月地进行下去。”

在斐波那契写出他的论著之前,印度学者已经讨论过 $\langle F_n\rangle$ 序列。他们对于由一个节拍和 2 个节拍的音符或音节形成的韵律模式感兴趣。这样有 n 个节拍的韵律个数合在一起是 F_{n+1};因此 Gopāla(1135 年以前)和 Hemacandra(大约 1150 年)都明确提到数 1,2,3,5,8,13,21,…。[请见 P. Singh, *Historia Math.* **12** (1985),229 ~ 244;也请参见习题

4.5.3-32。]

同一个序列也出现在 Johannes Kepler(开普勒)1601 年的著作中,他也在思考自己周围见到的这些数[J. Kepler, *The Six-Cornered Snowflake* (Oxford: Clarendon Press, 1966), 21]。大概是由于类似兔子问题原来的假设这样的原因,在自然界中斐波那契数经常被考察。[对此特别明晰的解释,参见 Conway 和 Guy, *The Book of Numbers* (New York: Copernicus, 1996), 113~126。]

有关 F_n 与算法之间的紧密关系的头一次提及,出现于 1837 年 É. Léger 使用斐波那契序列来研究欧几里得算法的效率时。他发现,如果算法 1.1E 中的数 m 和 n 不大于 F_k,步骤 E2 被执行的次数至多是 $k-1$。这是斐波那契序列的头一个实际应用(参见定理 4.5.3F)。在 19 世纪 70 年代,数学家 É. Lucas 得出关于斐波那契数很多有深远影响的结果。特别是,他利用它们来证明 39 位数字的数 $2^{127}-1$ 是素数。Lucas 给序列 $\langle F_n \rangle$ 取名"斐波那契数",这个名字一直沿用至今。

在 1.2.1 小节我们已经简略地考察了斐波那契序列(等式(3)及习题 4),其中我们发现,如果 n 是正整数且

$$\phi = \frac{1}{2}(1+\sqrt{5}) \tag{3}$$

则 $\phi^{n-2} \leq F_n \leq \phi^{n-1}$。我们很快就会看到这个量 ϕ 同斐波那契数有着非常密切的关系。

数 ϕ 本身有着非常有趣的历史。欧几里得把它叫做"端中比"(extreme and mean ratio);如果 A 对 B 的比是 ϕ 的话,则 A 对 B 的比等于 $A+B$ 对 A 之比。文艺复兴时期的作者把它叫做"神圣比例";而在上一世纪,人们普遍地把它叫做"黄金分割"。许多艺术家和作家都说,ϕ 对 1 之比是艺术上最令人喜爱的比例,而从计算机程序设计艺术的观点看来,他们的意见也是有道理的。关于 ϕ 的历史,请参见 H. S. M. Coxeter 的精彩论文"The Golden Section, Phyllotaxis, and Wythoff's Game", *Scripta Math.* **19** (1953), 135~143;也可见 Martin Gardner, *The 2nd Scientific American Book of Mathematical Puzzles and Diversions* 第 8 章(New York: Simon and Schuster, 1961)。George Markowsky 在 *College Math, J.* **23** (1992), 2~19 上已经揭穿了关于 ϕ 的若干流传的神话。事实是,F_{n+1}/F_n 之比趋于 ϕ 这一事实是由欧洲早期的计算能手 Simon Jacob 发现的,他死于 1564 年[参见 P. Schreiber, *Historia Math.* **22** (1995), 422~424。]

我们在这一小节里使用的记号是有点不太庄重的。在大量专门的数学著作里,F_n 改叫做 U_n 而 ϕ 改叫做 τ。我们的记号几乎普遍地为趣味性的数学著作(以及某些奇特的著作)所通用,而且它们正迅速地得到越来越广泛的使用。记号 ϕ 来自于古希腊艺术家 Phidias 的名字。据说,在他的雕塑中经常使用黄金分割。记号 F_n 是根据 *Fibonacci Quarterly* 杂志中的用法而使用的。在该杂志中,读者可以找到有关斐波那契序列的大量事实。关于 F_n 的经典著作的好的参考文献见 L. E. Dickson, *History of the Theory of Numbers* **1** (Carnegie Inst. of Washington, 1919),第 17 章。

斐波那契数满足许多有趣的恒等式,其中有些出现在本小节末尾的习题中。最普遍引用的关系之一,即开普勒 1608 年在他所写的一封信中提到但由 J. D. Cassini 首先发

表的[*Histoire Acad. Roy. Sci.* **1** (Paris, 1680), 201],就是

$$F_{n+1}F_{n-1} - F_n^2 = (-1)^n \tag{4}$$

这可用数学归纳法容易地证明。更巧妙的证明方法可由矩阵恒等式

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \tag{5}$$

的简单归纳法证明开始。然后我们取这个恒等式两边的行列式即得所证。

关系(4)说明 F_n 与 F_{n+1}是互素的,因为其任何公因子都将是$(-1)^n$ 的一个因子。

由定义(2)我们立即发现

$$F_{n+3} = F_{n+2} + F_{n+1} = 2F_{n+1} + F_n;\ F_{n+4} = 3F_{n+1} + 2F_n$$

而且一般说来,对于任意正整数 m,通过归纳法可以得到

$$F_{n+m} = F_mF_{n+1} + F_{m-1}F_n \tag{6}$$

如果我们在等式(6)中取 m 为 n 的倍数,则我们可归纳地推出

F_{nk}是F_n的倍数

因此,每第三个数是偶数,每第四个数是 3 的倍数,每第五个数是 5 的倍数,等等。

事实上,还有更多的事实成立。如果用 gcd(m,n)来表示 m 和 n 的最大公因子,则我们就有下列相当令人惊奇的定理:

定理 A(É. Lucas, 1876)　*一个数整除 F_m 和F_n 当且仅当它是F_d 的一个因子,其中 $d=\gcd(m,n)$;特别地,*

$$\gcd(F_m, F_n) = F_{\gcd(m,n)} \tag{7}$$

证明　通过使用欧几里得算法来证明这一结果。我们注意到由于等式(6),F_m 和 F_n 的任何公因子也是F_{n+m}的一个因子;反之,F_{n+m}和F_n 的任何公因子是F_mF_{n+1}的一个因子。由于 F_{n+1}和 F_n 互素,因此 F_{n+m}和 F_n 的一个公因子也整除 F_m。因此我们就证明了,对于任何数 d,

$$d\text{ 整除 }F_m\text{ 和 }F_n\text{ 当且仅当 }d\text{ 整除 }F_{m+n}\text{ 和 }F_n \tag{8}$$

我们现在将证明,使命题 8 成立且有 $F_0=0$ 的任何序列$\langle F_n\rangle$满足定理 A。

首先,显然命题(8)可通过对 k 的归纳法而被推广成以下规则:

$$d\text{ 整除 }F_m\text{ 和 }F_n\text{ 当且仅当 }d\text{ 整除 }F_{m+kn}\text{ 和 }F_n$$

其中 k 是任何非负整数。这一结果可以更精炼地表述为:

$$d\text{ 整除 }F_{m \bmod n}\text{ 和 }F_n\text{ 当且仅当 }d\text{ 整除 }F_m\text{ 和 }F_n \tag{9}$$

现在如果 r 是 m 除以 n 之后的余数,即如果 $r=m \bmod n$,则$\{F_m, F_n\}$的公因子是$\{F_n, F_r\}$的公因子。由此得出,当 m 和 n 改变时,贯穿算法 1.1E 的所有运算,$\{F_m, F_n\}$的公因子集合保持不变;最后,当 $r=0$ 时,公因子只不过是 $F_0=0$ 和 $F_{\gcd(m,n)}$的因子。▌

大多数涉及斐波那契数的重要结果都可以由 F_n 借助于 ϕ 的表示推演出来。我们现在就来进行推演。在以下的推导中我们将使用的方法极为重要,因此专长数学的读者应对之精心研究。在下一节里,我们还将详细地研究这同一方法。

我们通过建立无穷级数

$$G(z) = F_0 + F_1 z + F_2 z^2 + F_3 z^3 + F_4 z^4 + \cdots = z + z^2 + 2z^3 + 3z^4 + \cdots \tag{10}$$

开始。没有理由事先就预料这个无穷和是存在的或者函数 $G(z)$是全然值得研究的——但让我们对它抱着乐观的态度并且看看如果函数 $G(z)$确实存在的话,对它能做出什么结论。这样一种处理的优点在于,$G(z)$是一个同时表示整个斐波那契序列的单个的量;而且,如果我们发现 $G(z)$是一个“已知的”函数,则其系数就可以确定下来。我们称 $G(z)$是序列$\langle F_n\rangle$的*生成函数*。

现在我们可以如下来观察 $G(z)$:

$$zG(z) = F_0 z + F_1 z^2 + F_2 z^3 + F_3 z^4 + \cdots$$

$$z^2 G(z) = F_0 z^2 + F_1 z^3 + F_2 z^4 + \cdots$$

因此通过相减,得到

$$(1 - z - z^2)G(z) = F_0 + (F_1 - F_0)z + (F_2 - F_1 - F_0)z^2 + (F_3 - F_2 - F_1)z^3 + (F_4 - F_3 - F_2)z^4 + \cdots$$

根据 F_n 的定义,除第二项外所有的项都为零,因此这个表达式等于 z。因此如果 $G(z)$存在,则

$$G(z) = z/(1 - z - z^2) \tag{11}$$

事实上这个函数可被展开成 z 的一个无穷级数(泰勒级数);通过回溯,我们发现等式(11)的幂级数展开的系数必然是斐波那契数。

我们现在已经能处理 $G(z)$和找出关于斐波那契序列的更多结果了。分母 $1 - z - z^2$ 是有两个根$\frac{1}{2}(-1 \pm \sqrt{5})$的二次方程式。经少量计算后,可通过部分分式的方法把 $G(z)$展开成如下形式:

$$G(z) = \frac{1}{\sqrt{5}}\left(\frac{1}{1 - \phi z} - \frac{1}{1 - \hat{\phi} z}\right) \tag{12}$$

其中

$$\hat{\phi} = 1 - \phi = \frac{1}{2}(1 - \sqrt{5}) \tag{13}$$

量 $1/(1 - \phi z)$是无穷几何级数 $1 + \phi z + \phi^2 z^2 + \cdots$之和,因此我们有

$$G(z) = \frac{1}{\sqrt{5}}(1 + \phi z + \phi^2 z^2 + \cdots - 1 - \hat{\phi} z - \hat{\phi}^2 z^2 - \cdots)$$

现在来观察 z^n 的系数，它必然等于 F_n，因而我们求得

$$F_n = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n) \tag{14}$$

这是对斐波那契数的一个重要的封闭形式表示，它最先是在 18 世纪初发现的。(参见 D. Bernoulli, *Comment. Acad. Sci. Imp. Petrop* **3** (1728), 85 ~ 100, § 7；还参见 A. de Moivre, *Philos. Trans.* **32** (1922), 162 ~ 178。A. de Moivre 对于一般线性递归的解实际上是通过我们推导(14)的方法得到的。)

我们也可以只叙述等式(14)并用归纳法来证明它。但是上边稍微冗长的推导的目的在于说明，利用生成函数的重要方法，有可能首先发现这个等式。生成函数方法是解决广泛问题的有价值的技术。

许多事情都可由等式(14)证明出来。首先注意到 $\hat{\phi}$是一个负数($-0.61803\cdots$)，其绝对值小于 1，所以当 n 取充分大时，$\hat{\phi}^n$ 就变得很小。事实上 $\hat{\phi}^n/\sqrt{5}$总是足够地小，所以我们有

$$F_n = \phi^n/\sqrt{5}\ \textit{舍入到最接近的整数} \tag{15}$$

另一些结果可以从 $G(z)$直接得到，例如，

$$G(z)^2 = \frac{1}{5}\left(\frac{1}{(1-\phi z)^2} + \frac{1}{(1-\hat{\phi} z)^2} - \frac{2}{1-z-z^2}\right) \tag{16}$$

而且 $G(z)^2$ 中 z^n 的系数是$\sum_{k=0}^{n} F_k F_{n-k}$。因此我们就推出

$$\begin{aligned}\sum_{k=0}^{n} F_k F_{n-k} &= \frac{1}{5}((n+1)(\phi^n + \hat{\phi}^n) - 2F_{n+1}) = \\ &\frac{1}{5}((n+1)(F_n + 2F_{n-1}) - 2F_{n+1}) = \\ &\frac{1}{5}(n-1)F_n + \frac{2}{5}nF_{n-1}\end{aligned} \tag{17}$$

(本推演中的第二步由习题 11 的结果得出。)

习　题

1. [*10*] 斐波那契原来问题的答案是什么？一年之后会有多少对兔子？

▶ 2. [*20*] 鉴于等式(15)，F_{1000}的近似值是什么？(利用附录 A 中可找到的对数。)

3. [*25*] 写出在十进制下计算和打印 F_1 到 F_{1000}的一个计算机程序。(上题确定了必须处理的数的大小。)

▶ 4. [*14*] 求出使 $F_n = n$ 的所有 n。

5. [*20*] 求出使 $F_n = n^2$ 的所有 n。

6. [*HM10*] 证明等式(5)。

▶ 7. [*15*] 若 n 不是素数，则 F_n 也不是素数(仅有一个例外)。请证明这一命题和找出这一例

外。

8. [*15*] 在许多情况下,通过假定,对于所有整数 n,$F_{n+2}=F_{n+1}+F_n$,从而对于负的 n 来定义 F_n 是方便的。试探索这种可能性。F_{-1}等于什么? F_{-2}等于什么? 能否以简单的形式借助于 F_n 来表达 F_{-n}?

9. [*M20*] 利用习题 8 的约定,判定当允许下标取任意整数时,等式(4),(6),(14)和(15)是否仍然成立?

10. [*15*] $\phi^n/\sqrt{5}$是大于 F_n 还是小于 F_n?

11. [*20*] 证明对于所有整数 n,$\phi^n=F_n\phi+F_{n-1}$和 $\hat{\phi}^n=F_n\hat{\phi}+F_{n-1}$。

▶ 12. [*M26*] "二阶"斐波那契序列由下列规则定义:

$$\mathscr{F}_0=0,\ \mathscr{F}_1=1,\ \mathscr{F}_{n+2}=\mathscr{F}_{n+1}+\mathscr{F}_n+F_n$$

试借助于 F_n 和 F_{n+1}来表达 $\mathscr{F}_n$[提示:利用生成函数]。

▶ 13. [*M22*] 当 r,s 和 c 都是给定的常数时,借助于斐波那契数来表示以下序列:

a) $a_0=r,a_1=s;a_{n+2}=a_{n+1}+a_n,n\geqslant 0$。

b) $b_0=0,b_1=1;b_{n+2}=b_{n+1}+b_n+c,n\geqslant 0$。

14. [*M28*] 设 m 是一个固定的正整数。给定 $a_0=0,a_1=1;a_{n+2}=a_{n+1}+a_n+\binom{n}{m}$,其中 $n\geqslant 0$,试求 a_n。

15. [*M22*] 设 $f(n)$和 $g(n)$是任意的函数,且对于 $n\geqslant 0$,令

$$a_0=0,a_1=1;a_{n+2}=a_{n+1}+a_n+f(n)$$

$$b_0=0,b_1=1;b_{n+2}=b_{n+1}+b_n+g(n)$$

$$c_0=0,c_1=1;c_{n+2}=c_{n+1}+c_n+xf(n)+yg(n)$$

试借助于 x,y,a_n,b_n 和 F_n 表达 c_n。

▶ 16. [*M20*] 如果从正确的角度来看帕斯卡三角,则可以发现斐波那契数隐含地出现于其中,试证明下列的二项式系数之和是斐波那契数:

$$\sum_{k=0}^{n}\binom{n-k}{k}$$

17. [*M24*] 利用习题 8 的约定,证明等式(4)的下列推广:

$$F_{n+k}F_{m-k}-F_nF_m=(-1)^nF_{m-n-k}F_k$$

18. [*20*] $F_n^2+F_{n+1}^2$总是一个斐波那契数吗?

▶ 19. [*M27*] cos 36° 等于什么?

20. [*M16*] 借助于斐波那契数来表示$\sum_{k=0}^{n}F_k$。

21. [*M25*] $\sum_{k=0}^{n}F_kx^k$ 等于什么?

▶ 22. [*M20*] 证明$\sum_k\binom{n}{k}F_{m+k}$是一个斐波那契数。

23. [*M23*] 推广上一习题,证明$\sum_k\binom{n}{k}F_t^kF_{t-1}^{n-k}F_{m+k}$总是一个斐波那契数。

24. [*HM20*] 计算 $n\times n$ 行列式

$$\begin{pmatrix} 1 & -1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1 & -1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 1 \end{pmatrix}$$

25. [*M21*] 证明

$$2^n F_n = 2\sum_{k\text{奇数}} \binom{n}{k} 5^{(k-1)/2}$$

▶ 26. [*M20*] 利用上题证明，如果 p 为奇素数，则 $F_p \equiv 5^{(p-1)/2}$ (modulo p)。

27. [*M20*] 利用上题证明，如果 p 是不等于 5 的素数，则 F_{p-1}或 F_{p+1}之一是(但不都是) p 的倍数。

28. [*M21*] $F_{n+1} - \phi F_n$ 等于什么?

▶ 29. [*M23*](斐波那契系数) Édouard Lucas 以类似于二项式系数的方式定义了下列的量:

$$\binom{n}{k}_{\mathscr{F}} = \frac{F_n F_{n-1} \cdots F_{n-k+1}}{F_k F_{k-1} \cdots F_1} = \prod_{j=1}^{k} \left(\frac{F_{n-k+j}}{F_j} \right)$$

(a)对于 $0 \leqslant k \leqslant n \leqslant 6$ 试构造 $\binom{n}{k}_{\mathscr{F}}$ 的一个值表。(b)由于我们有

$$\binom{n}{k}_{\mathscr{F}} = F_{k-1}\binom{n-1}{k}_{\mathscr{F}} + F_{n-k+1}\binom{n-1}{k-1}_{\mathscr{F}}$$

试证明 $\binom{n}{k}_{\mathscr{F}}$ 总是一个整数。

▶ 30. [*M38*](D. Jarden, T. Motzkin) 斐波那契数 m 次方的序列满足一个递推关系，其中每一项都依赖于前边的 $m+1$ 项。证明

$$\sum_k \binom{m}{k}_{\mathscr{F}} (-1)^{\lceil (m-k)/2 \rceil} F_{n+k}^{m-1} = 0, \quad \text{若 } m > 0$$

例如，当 $m=3$ 时，我们得到恒等式 $F_n^2 - 2F_{n+1}^2 - 2F_{n+2}^2 + F_{n+3}^2 = 0$。

31. [*M20*] 证明 $F_{2n}\phi \bmod 1 = 1 - \phi^{-2n}$ 和 $F_{2n+1}\phi \bmod 1 = \phi^{-2n-1}$。

32. [*M24*] 一个斐波那契数除以另一个斐波那契数的余数是 ± (斐波那契数)，试证明，以 F_n 为模，

$$F_{mn+r} \equiv \begin{cases} F_r, & \text{如果 } m \bmod 4 = 0 \\ (-1)^{r+1} F_{n-r}, & \text{如果 } m \bmod 4 = 1 \\ (-1)^n F_r, & \text{如果 } m \bmod 4 = 2 \\ (-1)^{r+1+n} F_{n-r}, & \text{如果 } m \bmod 4 = 3 \end{cases}$$

33. [*HM24*] 给定 $z=\pi/2+i\ln\phi$,证明 $\sin nz/\sin z=i^{1-n}F_n$。

▶ 34. [*M24*](斐波那契数系) 令记号 $k\gg m$ 表示 $k\geqslant m+2$。证明每一个正整数 n 有一个惟一的表示 $n=F_{k_1}+F_{k_2}+\cdots+F_{k_r}$,其中 $k_1\gg k_2\gg\cdots\gg k_r\gg 0$。

35. [*M24*](ϕ 数系) 考虑利用基数 ϕ 并用 0 和 1 写成的实数表示,于是 $(100.1)_\phi=\phi^2+\phi^{-1}$。证明有无穷多的方法来表示数 1。例如 $1=(.11)_\phi=(.01111\cdots)_\phi$。但如果我们要求在表示中无两个相邻的 1 出现,且该表示不以无穷序列 01010101…结束,则每个非负数有惟一的表示。整数的表示如何?

▶ 36. [*M32*](斐波那契串) 令 S_1 = "a", S_2 = "b"和 $S_{n+2}=S_{n+1}S_n$, $n>0$;换言之 S_{n+2}是由把 S_n 放置于 S_{n+1}的右边而构成的。我们有 S_3 = "ba", S_4 = "bab", S_5 = "$babba$",等等。显然,S_n 有 F_n 个字母。试剖析 S_n 的性质。(何处出现重复的字母? 你能否预测 S_n 的第 k 个字母的值? b 的密度等于多少? 等等。)

▶ 37. [*M35*](R.E.Gaskell, M.J.Whinihan) 在下列游戏中两个人竞争:有一个含 n 个筹码的堆,头一个玩者可以从该堆中取走任意个筹码,但他不能取走所有的筹码。在那之后两个游戏者交替取走筹码,每个人可以取一个筹码或多个筹码,但不能取走多于对方上次取的个数的 2 倍,取走最后一个筹码的玩者获胜。(例如,假设 $n=11$;玩者 A 取走 3 个筹码,玩者 B 可以顶多取走 6 个筹码,他取一个。还剩 7 个筹码。玩者 A 可以取走 1 个或 2 个筹码,他取了 2 个筹码。玩者 B 可以顶多取走 4 个筹码,他取走 1 筹码,现在还剩 4 个筹码。玩者 A 现在取 1 个筹码;玩者 B 必须至少取走 1 个筹码,因此在下一轮中 A 获胜。)

如果开始时有 1000 个筹码,头一个玩者最好应取走多少个筹码?

38. [*35*] 写出描述并最优进行上题中游戏的计算机程序。

39. [*M24*] 给定 $a_0=0$, $a_1=1$ 和 $a_{n+2}=a_{n+1}+6a_n$, $n\geqslant 0$,试求出对于 a_n 的封闭形式的表达式。

40. [*M25*] 求解递推式

$$f(1)=0;\quad f(n)=\min_{0<k<n}\max(1+f(k),2+f(n-k)),\ n>1$$

▶ 41. [*M25*](Yuri Matiyasevich, 1990) 设 $f(x)=\lfloor x+\phi^{-1}\rfloor$。证明如果 $n=F_{k_1}+\cdots+F_{k_r}$是在习题 34 中的斐波那契数系下 n 的表示,则 $F_{k_1+1}+\cdots+F_{k_r+1}=f(\phi n)$。试求对于 $F_{k_1-1}+\cdots+F_{k_r-1}$ 的一个类似公式。

42. [*M26*](D.A.Klarner) 证明,如果 m 和 n 是非负整数,则有一个惟一的下标序列 $k_1\gg k_2\gg\cdots\gg k_r$ 使得

$$m=F_{k_1}+F_{k_2}+\cdots+F_{k_r},\quad n=F_{k_1+1}+F_{k_2+1}+\cdots+F_{k_r+1}$$

(k 可以为负,r 可以为零。)

1.2.9 生成函数

每当我们想要得到关于数列 $\langle a_n\rangle=a_0,a_1,a_2,\cdots$的信息时,可以借助于一个"参数" z 来建立一个无穷和

$$G(z)=a_0+a_1z+a_2z^2+\cdots=\sum_{n\geqslant 0}a_nz^n \tag{1}$$

然后尝试得到关于函数 G 的信息。这一函数是表示整个序列的单一的量；如果序列 $\langle a_n \rangle$ 已经归纳地定义（即如果 a_n 已经通过 $a_0, a_1, \cdots, a_{n-1}$ 加以定义），那么这是一个重要的优点。而且，假定等式(1)中的无穷和对于 z 的某个非零的值存在，通过利用微分技术，我们就可以从函数 $G(z)$ 来恢复 $a_0, a_1, \cdots$ 的值。

我们称 $G(z)$ 为序列 $a_0, a_1, a_2, \cdots$ 的生成函数。生成函数的利用，为我们引进了全新的技术领域。而且它广泛地增强了我们求解问题的能力。如同在上一节所提到的，A. de Moivre（棣莫弗）引进了生成函数，为的是解决一般的线性递归问题。斯特林把棣莫弗的理论推广到稍微复杂的递推式，他说明了怎样应用微分、积分以及算术运算[*Methodus Differentialis* (London:1730)，命题 15]。稍后几年，欧拉开始以若干新的途径来使用生成函数，如在他关于分划的一些论文中所见到的[*Commentarii Acad. Sci. Pet.* **13** (1741)，64 ~ 93；*Novi Comment Acad. Sci. Pet.* **3** (1750)，125 ~ 169]。Pierre S. Laplace（拉普拉斯）在他的名著*Théorie Analytique des Probabilités* (Paris:1812)中又进一步发展了这一技术。

无穷和(1)的收敛性问题有着一定的重要性，关于无穷级数理论的任何一本教材都将证明：

a)如果级数对于一个特殊的 z 值 $z = z_0$ 收敛，则它对于满足 $|z| < |z_0|$ 的所有 z 的值都收敛。

b)当且仅当序列 $\langle \sqrt[n]{|a_n|} \rangle$ 有界时对于某个 $z \neq 0$ 级数收敛。（如果这个条件不满足，我们能得到对于序列 $\langle a_n / n! \rangle$ 或对于某个其它有关序列的一个收敛的级数。）

另一方面，当我们使用生成函数来进行工作时，通常不必为级数的收敛性而担心，因为我们仅仅是探索对于某个问题的可能的解决途径。当我们无论通过什么途径发现解时，不论这个解如何草率，我们都可能独立地论证这个解的正确性。例如，在上一小节中，我们使用一个生成函数导致等式(14)；而一旦找到了这样一个等式，再用归纳法来证明它，那就是一件简单的事情，而且我也不必提及我们曾经利用生成函数来发现它这样一件事了。其次，可以证明，我们通过生成函数所做的大多数（如果不是全部的话）演算，都可严格地证明其正确性，而不必顾及级数的收敛性。例如，可参阅 E.T.Bell, *Trans. Amer. Math. Soc.* **25** (1923)，135 ~ 154；Ivan Niven, *AMM* **76** (1969)，871 ~ 889；Peter Henrici, *Applied and Computational Complex Analysis* **1** (Wiley, 1974)，第 1 章。

现在来研究对于生成函数所使用的主要技术。

A. 加法　如果 $G(z)$ 是对于 $\langle a_n \rangle = a_0, a_1, \cdots$ 的生成函数，而 $H(z)$ 是对于 $\langle b_n \rangle = b_0, b_1, \cdots$ 的生成函数。则 $\alpha G(z) + \beta H(z)$ 是对于 $\langle \alpha a_n + \beta b_n \rangle = \alpha a_0 + \beta b_0, \alpha a_1 + \beta b_1, \cdots$ 的生成函数：

$$\alpha \sum_{n \geq 0} a_n z^n + \beta \sum_{n \geq 0} b_n z^n = \sum (\alpha a_n + \beta b_n) z^n \tag{2}$$

B. 移位　如果 $G(z)$ 是对于 $\langle a_n \rangle = a_0, a_1, \cdots$ 的生成函数，则 $z^m G(z)$ 是对于 $\langle a_{n-m} \rangle = 0, \cdots, 0, a_0, a_1, \cdots$ 的生成函数：

$$z^m \sum_{n \geq 0} a_n z^n = \sum_{n \geq m} a_{n-m} z^n \tag{3}$$

如果我们认为对于 n 的任何负的值,都有 $a_n = 0$,则最后的那个求和可以扩展到所有 $n \geq 0$。

类似地,$(G(z) - a_0 - a_1 z - \cdots - a_{m-1} z^{m-1}) / z^m$ 是对于 $\langle a_{n+m} \rangle = a_m, a_{m+1}, \cdots$ 的生成函数:

$$z^{-m} \sum_{n \geq m} a_n z^n = \sum_{n \geq 0} a_{n+m} z^n \tag{4}$$

我们把运算 A 和 B 结合在一起,来解决前一小节中的斐波那契问题:$G(z)$是对于$\langle F_n \rangle$的生成函数,$zG(z)$是对于$\langle F_{n-1} \rangle$的,$z^2 G(z)$是对于$\langle F_{n-2} \rangle$的,以及$(1 - z - z^2) \cdot G(z)$是对于$\langle F_n - F_{n-1} - F_{n-2} \rangle$的。由于当 $n \geq 2$ 时,$F_n - F_{n-1} - F_{n-2}$为零,我们发现$(1 - z - z^2) G(z)$是一个多项式。类似地,给定任何线性递推序列,即满足$a_n = c_1 a_{n-1} + \cdots + c_m a_{n-m}$的序列,则生成函数将是可由$(1 - c_1 z - \cdots - c_m z^m)$整除的一个多项式。

让我们考虑所有生成函数当中最简单的情形:如果 $G(z)$是对于常数序列 1,1,1,…的生成函数,则 $zG(z)$生成 0,1,1,…,所以$(1 - z) G(z) = 1$。这便给出了虽然简单但却非常重要的公式

$$\frac{1}{1 - z} = 1 + z + z^2 + \cdots \tag{5}$$

C. 乘法 如果 $G(z)$是对于 $a_0, a_1, \cdots$的生成函数,而 $H(z)$是对于 $b_0, b_1, \cdots$的生成函数,则

$$G(z) H(z) = (a_0 + a_1 z + a_2 z^2 + \cdots)(b_0 + b_1 z + b_2 z^2 + \cdots) =$$
$$= (a_0 b_0) + (a_0 b_1 + a_1 b_0) z + (a_0 b_2 + a_1 b_1 + a_2 b_0) z^2 + \cdots$$

于是 $G(z) H(z)$是对于序列 $c_0, c_1, \cdots$的生成函数,其中

$$c_n = \sum_{k=0}^{n} a_k b_{n-k} \tag{6}$$

等式(3)是这个等式的一个非常特殊的情况。当每个 b_n 都等于 1 时出现另一个重要特殊情况:

$$\frac{1}{1 - z} G(z) = a_0 + (a_0 + a_1) z + (a_0 + a_1 + a_2) z^2 + \cdots \tag{7}$$

这里我们有原来序列之和的生成函数。

由(6)得出三个函数的乘积的规则;$F(z) G(z) H(z)$生成 $d_0, d_1, d_2, \cdots$,其中

$$d_n = \sum_{\substack{i,j,k \geq 0 \\ i+j+k=n}} a_i b_j c_k \tag{8}$$

对于任意个数的函数的乘积(当其有意义时)的一般规则是

$$\prod_{j\geq 0}\sum_{k\geq 0}a_{jk}z^k = \sum_{n\geq 0}z^n \sum_{\substack{k_0,k_1,\cdots\geq 0\\ k_0+k_1+\cdots=n}} a_{0k_0}a_{1k_1}\cdots \tag{9}$$

当对于某个序列的递推关系涉及二项式系数时我们通常要得到由

$$c_n = \sum_k \binom{n}{k} a_k b_{n-k} \tag{10}$$

定义的一个序列 $c_0, c_1, \cdots$的生成函数。在这种情况下通常使用对于序列$\langle a_n/n!\rangle$,$\langle b_n/n!\rangle$,$\langle c_n/n!\rangle$的生成函数要更好些,因为我们有

$$\left(\frac{a_0}{0!}+\frac{a_1}{1!}z+\frac{a_2}{2!}z^2+\cdots\right)\left(\frac{b_0}{0!}+\frac{b_1}{1!}z+\frac{b_2}{2!}z^2+\cdots\right)=$$

$$\left(\frac{c_0}{0!}+\frac{c_1}{1!}z+\frac{c_2}{2!}z^2+\cdots\right) \tag{11}$$

其中 c_n 是由等式(10)给出的。

D. z 的改变　显然 $G(cz)$是对于序列 $a_0, ca_1, c^2a_2, \cdots$的生成函数。作为一个特殊情况,对于 $1, c, c^2, c^3, \cdots$的生成函数是 $1/(1-cz)$。

为抽取一个级数的交替的项,有一个熟知的技巧:

$$\begin{aligned}\frac{1}{2}(G(z)+G(-z)) &= a_0 + a_2z^2 + a_4z^4 + \cdots\\ \frac{1}{2}(G(z)-G(-z)) &= \quad a_1z + a_3z^3 + a_5z^5 + \cdots\end{aligned} \tag{12}$$

利用 1 的复数根,我们可以把这个思想加以推广并且每隔 m 项来进行抽取,令 $\omega = e^{2\pi i/m} = \cos(2\pi/m) + i\sin(2\pi/m)$,我们有

$$\sum_{n \bmod m = r} a_n z^n = \frac{1}{m}\sum_{0\leq k<m}\omega^{-kr}G(\omega^k z),\quad 0\leq r<m \tag{13}$$

(参见习题 14。)例如,如果 $m=3$ 和 $r=1$,我们有 $\omega = -\frac{1}{2}+\frac{\sqrt{3}}{2}i$,此即 1 的复立方根;由此得出

$$a_1z + a_4z^4 + a_7z^7 + \cdots = \frac{1}{3}(G(z) + \omega^{-1}G(\omega z) + \omega^{-2}G(\omega^2 z))$$

E. 微分和积分　微积分的技术给了我们进一步的运算。如果 $G(z)$是由等式(1)给出的,则其导数为

$$G'(z) = a_1 + 2a_2z + 3a_3z^2 + \cdots = \sum_{k\geq 0}(k+1)a_{k+1}z^k \tag{14}$$

对序列$\langle na_n\rangle$的生成函数是 $zG'(z)$。因此我们可以通过处理生成函数而把一个序列的第 n 项同 n 的一个多项式组合起来。

逆转这个过程,积分给了我们另一个有用的操作:

$$\int_0^z G(t)\mathrm{d}t = a_0 z + \frac{1}{2}a_1 z^2 + \frac{1}{3}a_2 z^3 + \cdots = \sum_{k\geqslant 1}\frac{1}{k}a_{k-1}z^k \tag{15}$$

作为特殊情况,我们有(5)的导数和积分:

$$\frac{1}{(1-z)^2} = 1 + 2z + 3z^2 + \cdots = \sum_{k\geqslant 0}(k+1)z^k \tag{16}$$

$$\ln\frac{1}{1-z} = z + \frac{1}{2}z^2 + \frac{1}{3}z^3 + \cdots = \sum_{k\geqslant 1}\frac{1}{k}z^k \tag{17}$$

我们可以把第二个公式同等式(7)组合在一起来得到对于调和数的生成函数:

$$\frac{1}{1-z}\ln\frac{1}{1-z} = z + \frac{3}{2}z^2 + \frac{11}{6}z^3 + \cdots = \sum_{k\geqslant 0}H_k z^k \tag{18}$$

F. 已知的生成函数 只要有可能确定一个函数的幂级数展开,就意味着我们已经找到了对于一个特殊序列的生成函数。一经与上边所描述的运算相关联,这些特殊函数可以是十分有用的。下面列出的是最重要的幂级数展开。

i) 二项式定理

$$(1+z)^r = 1 + rz + \frac{r(r-1)}{2}z^2 + \cdots = \sum_{k\geqslant 0}\binom{r}{k}z^k \tag{19}$$

当 r 是一个负整数时,我们得到在等式(5)和(16)中已经反映出来的一个特殊情况:

$$\frac{1}{(1-z)^{n+1}} = \sum_{k\geqslant 0}\binom{-n-1}{k}(-z)^k = \sum_{k\geqslant 0}\binom{n+k}{n}z^k \tag{20}$$

还有一个推广,其证明在习题 1.2.6-25 中给出:如果 x 是求解方程 $x^{t+1} = x^t + z$ 的 z 的连续函数,其中当 $z=0$ 时,$x=1$,则

$$x^r = 1 + rz + \frac{r(r-2t-1)}{2}z^2 + \cdots = \sum_{k\geqslant 0}\binom{r-kt}{k}\frac{r}{r-kt}z^k \tag{21}$$

ii) 幂级数

$$\exp z = e^z = 1 + z + \frac{1}{2!}z^2 + \cdots = \sum_{k\geqslant 0}\frac{1}{k!}z^k \tag{22}$$

一般地说,我们有下列涉及斯特林数的公式:

$$(e^z - 1)^n = z^n + \frac{1}{n+1}\left\{{n+1 \atop n}\right\}z^{n+1} + \cdots = n!\sum_k\left\{{k \atop n}\right\}z^k/k! \tag{23}$$

iii) 对数级数(参见(17)和(18))

$$\ln(1+z) = z - \frac{1}{2}z^2 + \frac{1}{3}z^3 + \cdots = \sum_{k\geqslant 1}\frac{(-1)^{k-1}}{k}z^k \tag{24}$$

$$\frac{1}{(1-z)^{m+1}}\ln\left(\frac{1}{1-z}\right)=\sum_{k\geq 1}(H_{m+k}-H_m)\binom{m+k}{k}z^k \tag{25}$$

如同在(23)中那样,斯特林数给了我们更一般的等式:

$$\left(\ln\frac{1}{1-z}\right)^n=z^n+\frac{1}{n+1}\left[{n+1 \atop n}\right]z^{n+1}+\cdots=n!\sum_k\left[{k \atop n}\right]z^k/k! \tag{26}$$

进一步的推广,包括许多调和数的和,出现于 D.A.Zave 的文章中,见*Inf. Proc. Letters* **5** (1976),75~77;J.Spieß, *Math. Comp.* **55** (1990),839~863。

iv) 其他

$$z(z+1)\cdots(z+n-1)=\sum_k\left[{n \atop k}\right]z^k \tag{27}$$

$$\frac{z^n}{(1-z)(1-2z)\cdots(1-nz)}=\sum_k\left\{{k \atop n}\right\}z^k \tag{28}$$

$$\frac{z}{e^z-1}=1-\frac{1}{2}z+\frac{1}{12}z^2+\cdots=\sum_{k\geq 0}\frac{B_kz^k}{k!} \tag{29}$$

出现在上边最后一个公式的系数 B_k 是伯努利数。在 1.2.11.1 小节还将对它们作进一步的考察。附录 A 列出了伯努利数表。

类似于等式(21),下面的等式将在习题 2.3.4.4-29 中证明:

$$x^r=1+rz+\frac{r(r+2t)}{2}z^2+\cdots=\sum_{k\geq 0}\frac{r(r+kt)^{k-1}}{k!}z^k \tag{30}$$

如果 x 是求解方程 $x=e^{zx^t}$ 的 z 的连续函数,其中当 $z=0$ 时 $x=1$。在习题 4.7-22 中还将讨论(21)和(30)的重要推广。

G. 抽出一个系数　对于 $G(z)$中的 z^n 的系数使用记号

$$[z^n]G(z) \tag{31}$$

通常是方便的。例如,如果 $G(z)$是(1)中的生成函数,我们有$[z^n]G(z)=a_n$ 和 $[z^n]G(z)/(1-z)=\sum_{k=0}^n a_k$。在复变函数论中最基本的结果之一是柯西的一个公式 [*Exercises de Math.* **1** (1826),95~113 = *Œuvres* (2) **6**,124~145,等式(11)],通过这个公式,如果对于 $z=z_0$ 和 $0<r<|z_0|$,$G(z)$收敛,我们可以借助下列曲线积分来抽取任何想要的系数:

$$[z^n]G(z)=\frac{1}{2\pi i}\oint_{|z|=r}\frac{G(z)dz}{z^{n+1}} \tag{32}$$

基本的思想为,当这个积分是

$$\int_{-\pi}^{\pi}(re^{i\theta})^{-1}d(re^{i\theta})=i\int_{-\pi}^{\pi}d\theta=2\pi i$$

时，除 $m=-1$ 外，对于所有整数 m，$\oint_{|z|=r} z^m dz$ 为零。当要研究一个系数的近似值时，等式(32)是很重要的。

我们通过转回到在 1.2.3 小节里还仅部分地解决的问题来作为本小节的结束。在等式 1.2.3-(13)和习题 1.2.3-29 中我们看到

$$\sum_{1\leq i\leq j\leq n} x_i x_j = \frac{1}{2}\Big(\sum_{k=1}^{n} x_k\Big)^2 + \frac{1}{2}\Big(\sum_{k=1}^{n} x_k^2\Big)$$

$$\sum_{1\leq i\leq j\leq k\leq n} x_i x_j x_k = \frac{1}{6}\Big(\sum_{k=1}^{n} x_k\Big)^3 + \frac{1}{2}\Big(\sum_{k=1}^{n} x_k\Big)\Big(\sum_{k=1}^{n} x_k^2\Big) + \frac{1}{3}\Big(\sum_{k=1}^{n} x_k^3\Big)$$

一般来说，假设有 n 个数 $x_1, x_2, \cdots, x_n$，我们要求和数

$$h_m = \sum_{1\leq j_1\leq\cdots\leq j_m\leq n} x_{j_1}\cdots x_{j_m} \tag{33}$$

如果可能的话，要求借助于 $S_1, S_2, \cdots, S_m$ 来表达这个和，其中

$$S_j = \sum_{k=1}^{n} x_k^j \tag{34}$$

是 j 次幂的和数。利用这个紧凑的记法，上述公式变成

$$h_2 = \frac{1}{2}S_1^2 + \frac{1}{2}S_2; \quad h_3 = \frac{1}{6}S_1^3 + \frac{1}{2}S_1S_2 + \frac{1}{3}S_3$$

通过建立生成函数

$$G(z) = 1 + h_1 z + h_2 z^2 + \cdots = \sum_{k\geq 0} h_k z^k \tag{35}$$

我们可以解决这个问题。按照我们对级数实施乘法的规则，我们求得

$$G(z) = (1 + x_1 z + x_1^2 z^2 + \cdots)(1 + x_2 z + x_2^2 z^2 + \cdots)\cdots(1 + x_n z + x_n^2 z^2 + \cdots) =$$

$$\frac{1}{(1-x_1 z)(1-x_2 z)\cdots(1-x_n z)} \tag{36}$$

所以 $G(z)$ 是一个多项式的倒数。对一个乘积取对数通常是有利的，我们从(17)求得

$$\ln G(z) = \ln\frac{1}{1-x_1 z} + \cdots + \ln\frac{1}{1-x_n z} =$$

$$\Big(\sum_{k\geq 1}\frac{x_1^k z^k}{k}\Big) + \cdots + \Big(\sum_{k\geq 1}\frac{x_n^k z^k}{k}\Big) = \sum_{k\geq 1}\frac{S_k z^k}{k} \tag{37}$$

现在 $\ln G(z)$ 已经借助于诸 S 来表达了；因此为得到问题的答案，我们必须做的是，再次借助于(22)和(9)来计算 $G(z)$ 的幂级数展开：

$$G(z) = e^{\ln G(z)} = \exp\Big(\sum_{k\geq 1}\frac{S_k z^k}{k}\Big) = \prod_{k\geq 1} e^{S_k z^k/k} =$$

$$\left(1+S_1z+\frac{S_1^2z^2}{2!}+\cdots\right)\left(1+\frac{S_2z^2}{2}+\frac{S_2^2z^4}{2^2\cdot 2!}+\cdots\right)\cdots=$$

$$\sum_{m\geq 0}\left(\sum_{\substack{k_1,k_2,\cdots,k_m\geq 0\\ k_1+2k_2+\cdots+mk_m=m}}\frac{S_1^{k_1}}{1^{k_1}k_1!}\frac{S_2^{k_2}}{2^{k_2}k_2!}\cdots\frac{S_m^{k_m}}{m^{k_m}k_m!}\right)z^m \tag{38}$$

带圆括号的量是 h_m。在对它作了过细的考察后,这个使人敬畏的和实际上并不复杂。对于一个特定 m 值的项数是 $p(m)$,即 m 的分划数(1.2.1 小节)。例如,12 的一个分划是

$$12=5+2+2+2+1$$

这对应于方程 $k_1+2k_2+\cdots+12k_{12}=12$ 的一个解,其中 k_j 是在分划中 j 的个数。在我们的例子中,$k_1=1,k_2=3,k_5=1$,其它的 k 都为零。所以我们得到项

$$\frac{S_1}{1^1 1!}\frac{S_2^3}{2^3 3!}\frac{S_5}{5^1 1!}=\frac{1}{240}S_1S_2^3S_5$$

作为 h_{12}的表达式的一部分。通过对(37)求微分,不难导出递推式

$$h_n=\frac{1}{n}(S_1h_{n-1}+S_2h_{n-2}+\cdots+S_nh_0),\quad n\geq 1 \tag{39}$$

G. Pólya 已经给出了关于生成函数的应用的有趣介绍,"On picture writing", *AMM* **63** (1956),689～697;在*CMath* 第 7 章,给出了他的方法的继续。也可参见 H.S.Wilf 所著的书 *generatingfunctionology*, 第 2 版(Academic Press,1994)。

A generating function is a clothesline
on which we hang up a sequence of numbers for display.
一个生成函数就是一根晾衣绳,
我们把一个数列挂在上面供人看。
——H.S.WILF (1989)

习　题

1. [*M12*] 什么是对于序列 2,5,13,35,$\cdots=\langle 2^n+3^n\rangle$的生成函数?

▶ 2. [*M13*] 证明等式(11)。

3. [*HM21*] 求对$\langle H_n\rangle$的生成函数(18)的微分,并且把它同对$\langle \sum_{k=0}^n H_k\rangle$的生成函数作比较。你能推导出什么关系来?

4. [*M01*] 说明为什么等式(19)是等式(21)的一个特殊情况。

5. [*M20*] 对 n 用归纳法,证明等式(23)。

▶ 6. [*HM15*] 求出对于

$$\left\langle\sum_{0<k<n}\frac{1}{k(n-k)}\right\rangle$$

的生成函数,对它取微分并借助于调和数来表达其系数。

7. [*M15*] 验证导出等式(38)的所有步骤。

8. [*M23*] 求出对于 $p(n)$,即 n 的分划数的生成函数。

9. [*M11*] 在等式(34)和(35)的表达式之下,借助于 S_1, S_2, S_3 和 S_4 来表达,h_4 是什么?

▶10. [*M25*] 初等对称函数由公式

$$a_m = \sum_{1 \le j_1 < \cdots < j_m \le n} x_{j_1} \cdots x_{j_m}$$

定义。(这同等式(33)的 h_m 是一样的,但这不允许取相同的下标。)求出对于 a_m 的生成函数,并借助于等式(34)中的 S_j 来表达 a_m。写出对于 a_1, a_2, a_3 和 a_4 的公式。

▶11. [*M25*] 等式(39)也可用来借助于 h 表达诸 S:我们求得 $S_1 = h_1$, $S_2 = 2h_2 - h_1^2$, $S_3 = 3h_3 - 3h_1h_2 + h_1^3$,等等。当 $k_1 + 2k_2 + \cdots + mk_m = m$ 时,在 S_m 的这个表示中 $h_1^{k_1} h_2^{k_2} \cdots h_m^{k_m}$ 的系数是什么?

▶12. [*M20*] 假设我们对于 $m, n = 0, 1, \cdots$ 有一个双下标的序列 a_{mn};说明这一双下标序列怎样能通过一个双变量的生成函数来表示,并确定对于序列 $a_{mn} = \binom{n}{m}$ 的生成函数。

13. [*HM22*] 一个函数 $f(x)$ 的拉普拉斯变换是函数

$$\mathrm{L}f(s) = \int_0^\infty \mathrm{e}^{-st} f(t)\,\mathrm{d}t$$

给定无穷序列 $a_0, a_1, a_2, \cdots$,它具有收敛的生成函数。令 $f(x)$ 是阶梯函数 $\sum_k a_k [0 \le k \le x]$。试借助对于这个序列的生成函数 G 来表达 $f(x)$ 的拉普拉斯变换。

14. [*HM21*] 证明等式(13)。

15. [*M28*] 通过考虑 $H(w) = \sum_{n \ge 0} G_n(z) w^n$,试求生成函数

$$G_n(z) = \sum_{k=0}^{n} \binom{n-k}{k} z^k = \sum_{k=0}^{n} \binom{2k-n-1}{k} (-z)^k$$

16. [*M22*] 给出对于生成函数 $G_{nr}(z) = \sum_k a_{nkr} z^k$ 的一个简单公式,其中 a_{nkr} 是从 n 个对象中选出 k 个的方法种数,但附加上每个对象至多可选 r 次的条件。(如果 $r = 1$ 我们有 $\binom{n}{k}$ 种方法,而如果 $r \ge k$,如同在习题 1.2.6-60 一样,我们有允许重复的组合的数目。)

17. [*M25*] 如果把函数 $1/(1-z)^w$ 展开成关于 z 和 w 的双重幂级数,则其系数是什么?

▶18. [*M25*] 给定正整数 n 和 r,求下列和数值的简单公式:(a) $\sum_{1 \le k_1 < k_2 < \cdots < k_r \le n} k_1 k_2 \cdots k_r$;(b) $\sum_{1 \le k_1 \le k_2 \le \cdots \le k_r \le n} k_1 k_2 \cdots k_r$。(例如,当 $n = 3$ 和 $r = 2$ 时,这些和分别是 $1 \cdot 2 + 1 \cdot 3 + 2 \cdot 3$ 和 $1 \cdot 1 + 1 \cdot 2 + 1 \cdot 3 + 2 \cdot 2 + 2 \cdot 3 + 3 \cdot 3$。)

19. [*HM32*](F. Gauss, 1812) 下列无穷级数之和是众所熟知的:

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots = \ln 2; \quad 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots = \frac{\pi}{4}$$

$$1 - \frac{1}{4} + \frac{1}{7} - \frac{1}{10} + \cdots = \frac{\pi\sqrt{3}}{9} + \frac{1}{3}\ln 2$$

利用在习题 1.2.7-24 的答案中找到的定义

$$H_x = \sum_{n\geq 1}\left(\frac{1}{n} - \frac{1}{n+x}\right)$$

这些级数可分别写成

$$1 - \frac{1}{2}H_{1/2};\quad \frac{2}{3} - \frac{1}{4}H_{1/4} + \frac{1}{4}H_{3/4};\quad \frac{3}{4} - \frac{1}{6}H_{1/6} + \frac{1}{6}H_{2/3}$$

试证明,一般说来,当 p 和 q 是整数且 $0<p<q$ 时,$H_{p/q}$有值

$$\frac{q}{p} - \frac{\pi}{2}\cot\frac{p}{q}\pi - \ln 2q + 2\sum_{0<k<q/2}\cos\frac{2pk}{q}\pi\cdot\ln\sin\frac{k}{q}\pi$$

[提示:由阿贝尔的极限定理,这个和是

$$\lim_{x\to 1-}\sum_{n\geq 1}\left(\frac{1}{n} - \frac{1}{n+p/q}\right)x^{p+nq}$$

利用等式(13),通过使这一极限可以加以计算的方式表达这一幂级数。]

20. [*M21*] 对于$\sum_{n\geq 0}n^m z^n = \sum_{k=0}^{m}c_{mk}z^k/(1-z)^{k+1}$,系数 c_{mk}是什么?

21. [*HM30*] 建立对于序列$\langle n!\rangle$的生成函数并研究这个函数的性质。

22. [*M21*] 试求生成函数 $G(z)$,对于它有

$$[z^n]G(z) = \sum_{k_0+2k_1+4k_2+8k_3+\cdots=n}\binom{r}{k_0}\binom{r}{k_1}\binom{r}{k_2}\binom{r}{k_3}\cdots$$

23. [*M33*](L. Carlitz) (a) 证明对于所有整数 $m\geq 1$,存在多项式 $f_m(z_1,\cdots,z_m)$和 $g_m(z_1,\cdots,z_m)$使得公式

$$\sum_{k_1,\cdots,k_m\geq 0}\binom{r}{n-k_1}\binom{k_1}{n-k_2}\cdots\binom{k_{m-1}}{n-k_m}z_1^{k_1}\cdots z_m^{k_m} = f_m(z_1,\cdots,z_m)^{n-r}g_m(z_1,\cdots,z_m)^r$$

对于所有整数 $n\geq r\geq 0$ 是一个恒等式。

(b) 推广习题 15,借助于上边(a)中的函数 f_m 和 g_m 求对于和

$$S_n(z_1,\cdots,z_m) = \sum_{k_1,\cdots,k_m\geq 0}\binom{k_1}{n-k_2}\binom{k_2}{n-k_3}\cdots\binom{k_m}{n-k_1}z_1^{k_1}\cdots z_m^{k_m}$$

的一个封闭形式。

(c) 当 $z_1=\cdots=z_m=z$ 时,求对于 $S_n(z_1,\cdots,z_m)$的一个简单表达式。

24. [*M22*] 证明,如果 $G(z)$是任意生成函数,我们有

$$\sum_k\binom{m}{k}[z^{n-k}]G(z)^k = [z^n](1+zG(z))^m$$

当 $G(z)$是(a)$1/(1-z)$;(b)$(e^z-1)/z$ 时,计算这个恒等式的两边。

▶ 25. [*M23*] 通过简化等价公式$\sum_k[w^k](1-2w)^n[z^{n-k}](1+z)^{2n-2k}$,计算和

$$\sum_k\binom{n}{k}\binom{2n-2k}{n-k}(-2)^k$$

26. [*M40*] 探索记号(31)的一个推广,按照这个推广,比如,当 $G(z)$是由(1)给出的时,我们

可以写$[z^2-2z^5]G(z)=a_2-2a_5$。

1.2.10 一个算法的分析

现在让我们应用前几小节的某些技术研究一个典型的算法。

算法 M(求极大值) 给定 n 个元素 $X[1],X[2],\cdots,X[n]$,我们将求 m 和 j 使得 $m=X[j]=\max_{1\le i\le n}X[i]$,其中 j 是满足这个关系的最大下标。

M1.[初始化] 置 $j\leftarrow n,k\leftarrow n-1,m\leftarrow X[n]$。(在算法期间,我们将有 $m=X[j]=\max_{k<i\le n}X[i]$。)

M2.[全部试过了?] 如果 $k=0$,则算法终止。

M3.[比较] 如果 $X[k]\le m$,则转到 M5。

M4.[改变 m] 置 $j\leftarrow k,m\leftarrow X[k]$。(这个 m 的值是一个新的当前的极大值。)

M5.[减少 k] k 减 1 并返回 M2。▍

这个相当明显的算法可能显得如此平凡,因而我们无须自寻烦恼去详细分析它;但它实际上却很好地揭示了对于更复杂算法的研究方法。在计算机程序设计中,对算法的分析十分重要,因为通常对于一个特定的应用会有好多个算法可以利用,因而我们当然想要知道哪一个算法是最好的。

算法 M 要求固定数量的存储单元,因此我们将仅仅分析为实施它所需要的时间。为此,我们将计算执行每一步的次数(参见图 9):

步骤	次数
M1	1
M2	n
M3	$n-1$
M4	A
M5	$n-1$

知道每一步被执行的次数就给了我们确定在一个特定的计算机上的运行时间所需要的信息。

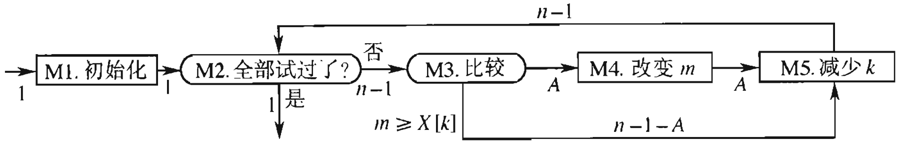


图 9,算法 M。箭头上的标号指出经过每一路径的次数。注意:必须满足"基尔霍夫第一定律":流入每个节点的量必须等于流出该点的量

在上边的次数表中,除了量 A 之外,每一个量都是我们所知道的。而量 A 是我们必须改变当前极大值的次数。为了完成整个分析,必须研究这个有趣的量 A。

这种分析通常由以下几个部分组成:求 A 的极小值(对于乐观的人们),A 的极大

值(对于悲观的人们),A 的平均值(对于讲究概率的人们),以及 A 的标准差(这是我们可以预期的值同平均值接近程度的数量指标)。

A 的极小值为零;当

$$X[n] = \max_{1\le k\le n} X[k]$$

时,即出现这种情况。极大值是 $n-1$;在

$$X[1] > X[2] > \cdots > X[n]$$

的情况下,即出现这种情况。

因此,平均值是在 0 与 $n-1$ 之间。它等于 $\frac{1}{2}n$ 吗? 它等于 $\sqrt{n}$ 吗? 为了回答这个问题,我们需要来定义所谓的"平均"意味着什么;而为了适当地定义平均,我们必须对输入数据 $X[1],X[2],\cdots,X[n]$ 的特征作某些假定。*我们将假定,$X[k]$ 都是不同的值,而且这些值的 $n!$ 个排列的每一个都是同等可能的。*(在大多数情况下这是一个合理的假定,但是如同本节末尾的习题中所示的那样,分析也可在另外的假定之下进行。)

算法 M 的性能不依赖于 $X[k]$ 的精确值,而仅仅涉及相对的次序。例如,当 $n=3$ 时,我们便假定,下列 6 个可能性中的每一个都是概率相同的:

情况	A 的值	情况	A 的值
$X[1]<X[2]<X[3]$	0	$X[2]<X[3]<X[1]$	1
$X[1]<X[3]<X[2]$	1	$X[3]<X[1]<X[2]$	1
$X[2]<X[1]<X[3]$	0	$X[3]<X[2]<X[1]$	2

当 $n=3$ 时,A 的平均值等于 $(0+1+0+1+1+2)/6=5/6$。

显然我们可以把 $X[1],X[2],\cdots,X[n]$ 取为在某个顺序下的 $1,2,\cdots,n$;在我们的假定之下,$n!$ 个排列中的每一个都是同样可能的。A 的值为 k 的概率将是

$$p_{nk} = (n\text{ 个对象的排列中满足 }A = k\text{ 的排列个数})/n! \tag{1}$$

例如,由上边的表,$p_{30}=\frac{1}{3}$,$p_{31}=\frac{1}{2}$,$p_{32}=\frac{1}{6}$。

像通常一样,平均("中"或"期望")值被定义为

$$A_n = \sum_k k p_{nk} \tag{2}$$

方差 V_n 被定义为 $(A-A_n)^2$ 的平均值;因此我们有

$$\begin{aligned} V_n &= \sum_k (k-A_n)^2 p_{nk} = \sum_k k^2 p_{nk} - 2A_n \sum_k k p_{nk} + A_n^2 \sum_k p_{nk} = \\ &\sum_k k^2 p_{nk} - 2A_n A_n + A_n^2 = \sum_k k^2 p_{nk} - A_n^2 \end{aligned} \tag{3}$$

最后,标准差 σ_n 被定义为 $\sqrt{V_n}$。

通过注意到,对于所有的 $r\ge 1$,A 落入它的均值的 $r\sigma_n$ 之外的概率小于 $1/r^2$,或许能最好地理解 σ_n 的意义。例如 $|A-A_n|>2\sigma_n$ 的概率 $<1/4$。(证明:令 p 是所述的概率,则若 $p>0$,$(A-A_n)^2$ 的平均值大于 $p(r\sigma_n)^2+(1-p)\cdot 0$;即 $V_n > pr^2V_n$。)通常把这叫做契比雪夫(Chebyshev)不等式,尽管如此,实际上它首先是由 J. Bienaymé 发现的

[*Comptes Rendus Acad. Sci.* **37** (Paris, 1853), 320 ~ 321]。

通过确定概率 p_{nk} 我们可以确定 A 的特性。归纳地来做这一点并不难：按照等式(1)，我们要计算满足 $A = k$ 的 n 个元素的排列的个数，令这个数为 $P_{nk} = n!\, p_{nk}$。

如同在 1.2.5 小节那样，考虑在 $\{1,2,\cdots,n\}$ 上的排列 $x_1\ x_2\ \cdots\ x_n$，如果 $x_1 = n$，A 的值比在 $x_2\ \cdots\ x_n$ 上所得到的值大 1；如果 $x_1 \neq n$，则 A 的值就和在 $x_2\ \cdots\ x_n$ 上的值完全一样。因此我们求得 $P_{nk} = P_{(n-1)(k-1)} + (n-1)P_{(n-1)k}$，或者等价地

$$p_{nk} = \frac{1}{n}p_{(n-1)(k-1)} + \frac{n-1}{n}p_{(n-1)k} \tag{4}$$

如果我们提供初始条件

$$p_{1k} = \delta_{0k}; \qquad p_{nk} = 0, \qquad \text{如果 } k < 0 \tag{5}$$

这个等式将确定 p_{nk}。

通过使用生成函数我们现在能得到关于量 p_{nk} 的信息，令

$$G_n(z) = p_{n0} + p_{n1}z + \cdots = \sum_k p_{nk}z^k \tag{6}$$

我们知道，$A \leqslant n-1$，因此对于 k 的很大的值，$p_{nk} = 0$；因此 $G_n(z)$ 实际上是一个多项式，尽管为方便起见已将其确定为无穷和。

由等式(5)我们有 $G_1(z) = 1$；而且由等式(4)我们有

$$G_n(z) = \frac{z}{n}G_{n-1}(z) + \frac{n-1}{n}G_{n-1}(z) = \frac{z+n-1}{n}G_{n-1}(z) \tag{7}$$

(读者应仔细研究等式(4)和等式(7)之间的关系。)我们现在看到

$$G_n(z) = \frac{z+n-1}{n}G_{n-1}(z) = \frac{z+n-1}{n}\,\frac{z+n-2}{n-1}G_{n-2}(z) = \cdots =$$

$$\frac{1}{n!}(z+n-1)(z+n-2)\cdots(z+1) =$$

$$\frac{1}{z+n}\binom{z+n}{n} \tag{8}$$

所以 $G_n(z)$ 实质上是一个二项式系数！

在上一小节中，这个函数已出现过，即等式 1.2.9-(27)，其中有

$$G_n(z) = \frac{1}{n!}\sum_k \left[{n \atop k}\right] z^{k-1}$$

因此 p_{nk} 可借助于斯特林数加以表达：

$$p_{nk} = \left[{n \atop k+1}\right] \Big/ n! \tag{9}$$

图 10 示出，当 $n = 12$ 时，p_{nk} 的近似大小。

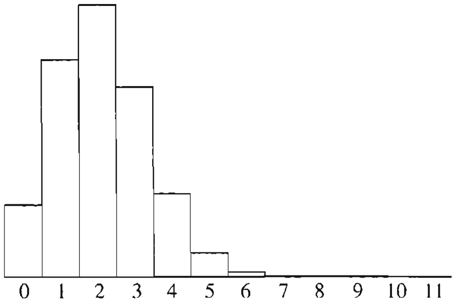


图 10　当 $n=12$ 时步骤 M4 的概率分布。均值是 58301/27720，近似于 2.10。方差近似于 1.54

现在我们所要做的就是把 p_{nk} 的这个值代入等式(2)和(3)中，并得到所想要的均值。但做比说难。事实上，能明确地确定概率 p_{nk} 是不寻常的；在大多数的问题中，我们将知道生成函数 $G_n(z)$，但关于实际的概率的具体值我们一无所知。重要的事实是：*由生成函数本身，我们能容易地确定均值和方差。*

为了了解这一点，假设我们有一个其系数表示概率的生成函数

$$G(z) = p_0 + p_1 z + p_2 z^2 + \cdots$$

这里 p_k 是某个事件有值 k 的概率，我们希望计算下列量：

$$\text{mean}(G) = \sum_k k p_k, \quad \text{var}(G) = \sum_k k^2 p_k - (\text{mean}(G))^2 \tag{10}$$

利用微分，不难发现计算方法。注意

$$G(1) = 1 \tag{11}$$

因为 $G(1) = p_0 + p_1 + p_2 + \cdots$ 是所有可能的概率之和。类似地，由于 $G'(z) = \sum_k k p_k z^{k-1}$，我们有

$$\text{mean}(G) = \sum_k k p_k = G'(1) \tag{12}$$

最后，我们再次应用微分而得到(参见习题 2)

$$\text{var}(G) = G''(1) + G'(1) - G'(1)^2 \tag{13}$$

等式(12)和(13)借助于生成函数给出了所求的均值和方差的表达式。

在当前的情况下，我们希望计算 $G'_n(1) = A_n$。由等式(7)，我们有

$$G'_n(z) = \frac{1}{n} G_{n-1}(z) + \frac{z+n-1}{n} G'_{n-1}(z)$$

$$G'_n(1) = \frac{1}{n} + G'_{n-1}(1)$$

由初始条件 $G'_1(1) = 0$，因此我们求得

$$A_n = G'_n(1) = H_n - 1$$

这就是所求的步骤 M4 被执行次数的平均数；当 n 很大时，它近似于 $\ln n$。[注：$A+1$ 的

第 r 次矩,即量 $\sum_k (k+1)^r p_{nk}$,是 $[z^n](1-z)^{-1}\sum_k \left\{ {r \atop k} \right\} \left(\ln \frac{1}{1-z}\right)^k$,而且它有 $(\ln n)^r$ 的近似值,参见 P.B.M.Roes,*CACM* **9** (1966),342。头一个研究 A 的分布的是 F.G.Foster 和 A.Stuart, *J.Roy.Stat.Soc.* **B16** (1954),1~22。]

我们可以类似地进行方差 V_n 的计算。但在此之前,且让我们叙述一个重要的简化。

定理 A *设 G 和 H 是满足 $G(1)=H(1)=1$ 的两个生成函数,如果以等式(12)和(13)定义量* mean(G)*和* var(G),*我们有*

$$\text{mean}(GH) = \text{mean}(G) + \text{mean}(H); \qquad \text{var}(GH) = \text{var}(G) + \text{var}(H) \tag{15}$$

我们将在后边证明这个定理。这个定理告诉我们:生成函数的积的均值和方差可以简化为一个和数。▌

设 $Q_n(z)=(z+n-1)/n$,我们有 $Q'_n(1)=1/n$, $Q''_n(1)=0$;因此

$$\text{mean}(Q_n) = \frac{1}{n}, \quad \text{var}(Q_n) = \frac{1}{n} - \frac{1}{n^2}$$

最后,由于 $G_n(z)=\prod_{k=2}^{n} Q_k(z)$,由此得出

$$\text{mean}(G_n) = \sum_{k=2}^{n} \text{mean}(Q_k) = \sum_{k=2}^{n} \frac{1}{k} = H_n - 1$$

$$\text{var}(G_n) = \sum_{k=2}^{n} \text{var}(Q_k) = \sum_{k=1}^{n} \left(\frac{1}{k} - \frac{1}{k^2}\right) = H_n - H_n^{(2)}$$

综合起来,我们已经得出所求的关于量 A 的统计

$$A = \left(\min 0, \quad \text{ave } H_n - 1, \quad \max n-1, \quad \text{dev } \sqrt{H_n - H_n^{(2)}}\right) \tag{16}$$

等式(16)中所用的符号将贯穿于本书用于描述其它概率量的统计特征。

我们已经完成了对于算法 M 的分析;在这个分析中出现的新的特征是引进了概率论。对于本书中的大多数应用来说,初等概率论就足够了:简单的计数技术以及已经给出的均值、方差和标准差的定义将回答我们想要问的大多数问题。更复杂的算法将帮助我们培养对概率进行熟练推理的能力。

让我们考虑某些简单的概率问题,以便获得使用这些方法的更多一点的实践。在所有的概率问题中,人们首先想到的问题是硬币投掷问题:假设我们投掷 n 次,而且在任何特定的投掷之后,正面朝上的概率是 p;问正面将出现的平均次数是多少? 标准差等于什么?

我们将认为硬币是有偏倚的;即不是假定 $p=\frac{1}{2}$。这就使问题更有趣些,而且其实每一个实际的硬币都是有偏倚的(否则我们就不可能区别这一面还是那一面了)。

如同前边那样进行,我们设 p_{nk} 为 k 次正面出现的概率,且令 $G_n(z)$ 为对应的生成函数。显然我们有

$$p_{nk} = pp_{(n-1)(k-1)} + qp_{(n-1)k} \tag{17}$$

其中 $q = 1 - p$ 是背面朝上的概率。如同前边一样，由等式(17)我们论证 $G_n(z) = (q + pz)G_{n-1}(z)$；而且，由明显的初始条件 $G_1(z) = q + pz$ 我们有

$$G_n(z) = (q + pz)^n \tag{18}$$

因此，根据定理 A，

$$\text{mean}(G_n) = n\ \text{mean}(G_1) = pn$$

$$\text{var}(G_n) = n\ \text{var}(G_1) = (p - p^2)n = pqn$$

关于正面数目，因此有

$$(\min 0,\quad \text{ave}\ pn,\quad \max n,\quad \text{dev}\ \sqrt{pqn}) \tag{19}$$

图 11 示出当 $p = \frac{3}{5}$，$n = 12$ 时，p_{nk}的一些值。当标准差与$\sqrt{n}$成正比，而且极大值与极小值间之差与 n 成正比时，我们可以认为关于平均的状态是“稳定的”。

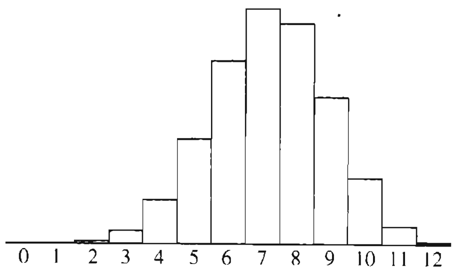


图 11　硬币投掷的概率分布：12 个独立的投掷且每次投掷成功机会等于$\frac{3}{5}$

让我们研究一个更简单的问题。假设在某个进程中，有相等的概率来得到值 1，2，…，n。对于这种情况的生成函数是

$$G(z) = \frac{1}{n}z + \frac{1}{n}z^2 + \cdots + \frac{1}{n}z^n = \frac{1}{n}\frac{z^{n+1} - z}{z - 1} \tag{20}$$

在经过稍微繁琐的计算之后，我们求得

$$G'(z) = \frac{nz^{n+1} - (n+1)z^n + 1}{z(z-1)^2}$$

$$G''(z) = \frac{n(n-1)z^{n+1} - 2(n+1)(n-1)z^n + n(n+1)z^{n-1} - 2}{n(z-1)^3}$$

现在为计算均值和方差，我们需要知道 $G'(1)$和 $G''(1)$。但当我们代入 $z = 1$ 时，用于表达这些量的形式简化为 0/0。这使得有必要来求当 z 趋于 1 时这个极限的值，而这是一个不平凡的任务。

幸而有一个简单得多的方法来做。由泰勒定理，我们有

$$G(1+z)=G(1)+G'(1)z+\frac{G''(1)}{2!}z^2+\cdots \tag{21}$$

因此只需要在(20)中,以 $z+1$ 来代替 z,我们就可得出系数

$$G(1+z)=\frac{1}{n}\frac{(1+z)^{n+1}-1-z}{z}=1+\frac{n+1}{2}z+\frac{(n+1)(n-1)}{6}z^2+\cdots$$

由此得出 $G'(1)=\frac{1}{2}(n+1)$, $G''(1)=\frac{1}{3}(n+1)(n-1)$,而且对于均匀分布的统计量是

$$\left(\min 1,\quad \text{ave}\,\frac{n+1}{2},\quad \max n,\quad \text{dev}\sqrt{\frac{(n+1)(n-1)}{12}}\right) \tag{22}$$

在此情况下,可以认为近似 $0.289n$ 的偏差是不稳定的。

通过证明定理 A 和把我们的概念同经典概率论关联起来,我们来结束本小节。假设 X 是仅取非负整数值的一个随机变量。其中 $X=k$ 的概率为 p_k,于是 $G(z)=p_0+p_1z+p_2z^2+\cdots$称做对于 X 的概率生成函数,而且量 $G(e^{it})=p_0+p_1e^{it}+p_2e^{2it}+\cdots$习惯上叫做这个分布的特征函数。由两个这样的生成函数的乘积给出的分布叫做这两个分布的卷积(convolution),而且它表示属于这些分布的两个独立随机变量的和。

一个随机量 X 的均值叫做它的期望值,并以 EX 记之。X 的方差则为 $EX^2-(EX)^2$。在这一记法下,X 的概率生成函数是 $G(z)=Ez^X$,即在 X 仅取非负整数的情况下 z^X 的期望值。类似地,如果 X 是一个取真值或假值的命题,利用 Iverson 的约定(等式 1.2.3-(16)),X 为真的概率是 $\Pr(X)=E[X]$。

均值和方差恰恰就是 Thiele 于 1903 年引进的两个所谓半不变量或累积量。半不变量 $\kappa_1,\kappa_2,\kappa_3,\cdots$由下列规则定义:

$$\frac{\kappa_1 t}{1!}+\frac{\kappa_2 t^2}{2!}+\frac{\kappa_3 t^3}{3!}+\cdots=\ln G(e^t) \tag{23}$$

我们有

$$\kappa_n=\frac{d^n}{dt^n}\ln G(e^t)\bigg|_{t=0}$$

特别是

$$\kappa_1=\frac{e^tG'(e^t)}{G(e^t)}\bigg|_{t=0}=G'(1)$$

因为 $G(1)=\sum_k p_k=1$,而且

$$\kappa_2=\frac{e^{2t}G''(e^t)}{G(e^t)}+\frac{e^tG'(e^t)}{G(e^t)}-\frac{e^{2t}G'(e^t)^2}{G(e^t)^2}\bigg|_{t=0}=G''(1)+G'(1)-G'(1)^2$$

由于半不变量是借助于一个生成函数的对数定义的,定理 A 因而是显然的,而且事实上,可以把它推广而应用于所有的半不变量上。

一个正态分布是这样一个分布,对于它,除均值和方差外,所有半不变量都为零。

在一个正态分布中,我们可对契比雪夫不等式作重大改进:一个正态分布的随机值同均值的差异小于标准差的概率是

$$\frac{1}{\sqrt{2\pi}}\int_{-1}^{+1}e^{-t^2/2}dt$$

即大约是 68.268949213709% 的时间。差异小于偏差的两倍的概率大约是 95.449973610364%的时间,而它少于标准差的三倍的概率大约是 99.730020393674%的时间。当 n 很大时,由等式(8)和(18)所确定的分布近似地是正态的(参看习题 13 和 14)。

我们通常需要知道一个随机变量是否不可能比它的均值大得很多或小得很多。两个极端简单但却强有力的公式,称为*尾不等式*,提供了对于这样一些概率的方便的估计。如果 X 有概率生成函数 $G(z)$,则

$$\Pr(X \leqslant r) \leqslant x^{-r}G(x), \qquad \text{对于 } 0 < x \leqslant 1 \tag{24}$$

$$\Pr(X \geqslant r) \leqslant x^{-r}G(x), \qquad \text{对于 } x \geqslant 1 \tag{25}$$

证明很容易:如果 $G(z) = p_0 + p_1 z + p_2 z^2 + \cdots$,我们有,当 $0 < x \leqslant 1$ 时,

$$\Pr(X \leqslant r) = p_0 + p_1 + \cdots + p_{\lfloor r\rfloor} \leqslant x^{-r}p_0 + x^{1-r}p_1 + \cdots + x^{\lfloor r\rfloor - r}p_{\lfloor r\rfloor} \leqslant x^{-r}G(x)$$

而当 $x \geqslant 1$ 时,

$$\Pr(X \geqslant r) = p_{\lceil r\rceil} + p_{\lceil r\rceil+1} + \cdots \leqslant x^{\lceil r\rceil - r}p_{\lceil r\rceil} + x^{\lceil r\rceil+1-r}p_{\lceil r\rceil+1} + \cdots \leqslant x^{-r}G(x)$$

通过选择极小化或近似地极小化(24)和(25)右边的 x 的值,我们通常得到相当接近于左边真正的尾概率的上界。

习题 21-23 说明了在若干重要情况下的尾不等式。这些不等式是由 A.N.Kolmogorov 在他的书*Grundbegriffe der Wahrscheinlichkeitsrechnung* (Springer,1933)首先指出的更一般原理的特殊情况:如果对于所有 $t \geqslant r$, $f(t) \geqslant s > 0$,则只要 $\mathrm{E}f(x)$存在,就有 $\Pr(X \geqslant r) \leqslant s^{-1}\mathrm{E}f(X)$。当 $f(t) = x^t$ 和 $s = x^r$ 时,我们便得到(25)。

习　题

1. [*10*] 由等式(4)和(5)确定 p_{n0}的值,并从算法 M 的观点来解释这一结果。

2. [*HM16*] 由等式(10)导出等式(13)。

3. [*M15*] 如果使用算法 M 来求 1000 个随机地排序的不同的项目,步骤 M4 被执行次数的极大值、极小值、平均值以及标准差等于什么?(给出这些量的小数近似值。)

4. [*M10*] 给出在硬币投掷的实验,即等式(17)中,对于 p_{nk}的值的一个明晰的封闭公式。

5. [*M13*] 什么是在图 11 中的分布的均值和标准差?

6. [*HM27*] 我们已经计算了重要的概率分布(8),(18),(20)的均值和方差。试问在这些情况的每一个中,第 3 半不变量 κ_3 是什么?

▶ 7. [*M27*] 在对于算法 M 的分析中,我们假定所有 $X[k]$都是不同的。设想我们代之以仅作

较弱的假定，即 $X[1],X[2],\cdots,X[n]$恰包含 m 个不同的值；在这个条件的限制下，此外的值是随机的。在此情况下 A 的概率分布是什么？

▶ 8. [*M20*] 假设每个 $X[k]$都是从 M 个不同元素的集合中随机地取的，使得对于 $X[1]$，$X[2],\cdots,X[n]$的 M^n 个可能的选择的每一个都是同等可能的。问所有 $X[k]$都将是不同的概率是什么？

9. [*M25*] 推广上题的结果来求在诸 X 当中，恰有 m 个不同的值的概率的公式。用斯特林数表示你的答案。

10. [*M20*] 把上边三题的结果组合在一起，得出在从M个对象的一个集合中随机地选择每个 X 的假定下，对于 $A=k$ 的概率的一个公式。

▶ 11. [*M15*] 如果我们把 $G(z)$变为 $F(z)=z^nG(z)$，一个分布的半不变量会发生什么情况？

12. [*HM21*] 当 $G(z)=p_0+p_1z+p_2z^2+\cdots$表示一个概率分布时，量 $M_n=\sum_k k^np_k$ 和 $m_n=\sum_k(k-M_1)^np_k$ 分别叫做"第 n 次矩"和"第 n 次中心矩"。证明 $G(e^t)=1+M_1t+M_2t^2/2!+\cdots$；然后使用 Arbogast 公式(习题 1.2.5-21)证明

$$\kappa_n=\sum_{\substack{k_1,k_2,\cdots,k_n\geq 0\\ k_1+2k_2+\cdots=n}}\frac{(-1)^{k_1+k_2+\cdots+k_n-1}n!(k_1+k_2+\cdots+k_n-1)!}{k_1!1!^{k_1}k_2!2!^{k_2}\cdots k_n!n!^{k_n}}M_1^{k_1}M_2^{k_2}\cdots M_n^{k_n}$$

特别是，$\kappa_1=M_1$，$\kappa_2=M_2-M_1^2$(如同我们已知的那样)，$\kappa_3=M_3-3M_1M_2+2M_1^3$，以及 $\kappa_4=M_4-4M_1M_3+12M_1^2M_2-3M_2^2-6M_1^4$。当 $n\geq 2$ 时，借助于中心矩 $m_2,m_3,\cdots$，对于 κ_n 的类似表达式是什么？

13. [*HM38*] 对带有均值 μ_n 和标准差 σ_n 的概率生成函数 $G_n(z)$序列，如果对于 t 的所有实数值，

$$\lim_{n\to\infty}e^{-it\mu_n/\sigma_n}G_n(e^{it/\sigma_n})=e^{-t^2/2}$$

则称该序列趋近正态分布。利用由等式(8)给出的那样的 $G_n(z)$，证明 $G_n(z)$趋近正态分布。

注：如同在这里所定义的一样，"趋近正态分布"，可被证明是等价于以下事实，即

$$\lim_{n\to\infty}\Pr\left(\frac{X_n-\mu_n}{\sigma_n}\leq x\right)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x}e^{-t^2/2}dt$$

其中 X_n 是一个随机量，它的概率由 $G_n(z)$所确定。这是 P. Lévy 重要的"连续性定理"的一个特殊情况。这个连续性定理是数学概率论的一个基本结果。Lévy 定理的证明尽管并非极其困难[例如，参见 B. V. Gnedenko 和 A. N. Kolmogorov 著的*Limit Distribution for Sums of Independent Random Variables*，钟开莱(K. L. Chung)译成英文(Reading, Mass.: Addison-Wesley, 1954)]*，但这里没必要扯那么远。

14. [*HM30*](A. de Moivre) 利用上题的约定，证明由(18)给出的二项式分布 $G_n(z)$趋近正态分布。

15. [*HM23*] 当某个量有值 k 的概率是 $e^{-\mu}(\mu^k/k!)$时，就说它有均值为 μ 的泊松分布。

a) 对于这类概率的生成函数是什么？

b) 半不变量的值是什么？

---

* 中译本由王寿仁译出，科学出版社，1955。——译者注

c) 证明当 $n\to\infty$ 时，具有均值 np 的泊松分布在习题 13 的意义下趋近正态分布。

16. [*M25*] 假设 X 是一个随机变量，其值在下列意义下是由 $g_1(z), g_2(z), \cdots, g_r(z)$ 生成的概率分布的混合，即它以概率 p_k 使用 $g_k(z)$，且 $p_1+p_2+\cdots+p_r=1$。问 X 的生成函数是什么？借助于 $g_1, g_2, \cdots, g_r$ 的均值与方差来表达 X 的均值与方差。

▶ 17. [*M27*] 设 $f(z)$ 和 $g(z)$ 是表示概率分布的生成函数。

a) 证明 $h(z)=g(f(z))$ 也是表示一个概率分布的生成函数。

b) 借助于 $f(z)$ 和 $g(z)$ 解释 $h(z)$ 的意义。(什么是由 $h(z)$ 的系数表示的概率的意义？)

c) 借助于 f 和 g 的均值和方差给出 h 的均值和方差的公式。

18. [*M28*] 假设在算法 M 中由 $X[1], X[2], \cdots, X[n]$ 所取的值恰含 k_1 个 1，k_2 个 2，…，k_n 个 n，其安排是随机的。(这里 $k_1+k_2+\cdots+k_n=n$。在正文中假定 $k_1=k_2=\cdots=k_n=1$。)试证明在这一推广的情况下，利用约定 $0/0=1$，生成函数(8)变成

$$\left(\frac{k_{n-1}z+k_n}{k_{n-1}+k_n}\right)\left(\frac{k_{n-2}z+k_{n-1}+k_n}{k_{n-2}+k_{n-1}+k_n}\right)\cdots\left(\frac{k_1z+k_2+\cdots+k_n}{k_1+k_2+\cdots+k_n}\right)$$

19. [*M21*] 如果对于 $1\leqslant j<k$，$a_k>a_j$，我们便说 a_k 是序列 $a_1a_2\cdots a_n$ 的*自左至右的极大值*。假设 $a_1a_2\cdots a_n$ 是 $\{1,2,\cdots n\}$ 的一个排列，并令 $b_1b_2\cdots b_n$ 是逆排列，使得当且仅当 $b_l=k$ 时，$a_k=l$。试证明 a_k 是 $a_1a_2\cdots a_n$ 的自左至右的极大值当且仅当 k 是 $b_1b_2\cdots b_n$ 的自右至左的极大值。

▶ 20. [*M22*] 假设我们要计算当 $b_1\leqslant b_2\leqslant\cdots\leqslant b_n$ 时的 $\max\{|a_1-b_1|, |a_2-b_2|, \cdots, |a_n-b_n|\}$，试证明只要计算 $\max\{m_L, m_R\}$ 即可，其中

$$m_L=\max\{a_k-b_k \mid a_k\text{ 是 }a_1a_2\cdots a_n\text{ 的自左至右的极大值}\}$$

$$m_R=\max\{b_k-a_k \mid a_k\text{ 是 }a_1a_2\cdots a_n\text{ 自右至左的极小值}\}$$

[因此如果诸 a 是在随机的次序之下，对之必须做减法的 k 的数目仅约为 $2\ln n$。]

▶ 21. [*HM21*] 设 x 是当一个随机的硬币被投掷 n 次而出现的是正面的次数，且有生成函数(18)。试利用(25)证明，当 $\epsilon\geqslant 0$ 时，

$$\Pr(X\geqslant n(p+\epsilon))\leqslant e^{-\epsilon^2n/(2q)}$$

并求得对于 $\Pr(X\leqslant n(p-\epsilon))$ 类似的估计。

▶ 22. [*HM22*] 假设 X 有生成函数 $(q_1+p_1z)(q_2+p_2z)\cdots(q_n+p_nz)$，其中对于 $1\leqslant k\leqslant n$ 有 $p_k+q_k=1$。令 $\mu=EX=p_1+p_2+\cdots+p_n$。(a)证明

$$\Pr(X\leqslant\mu r)\leqslant(r^{-r}e^{r-1})^{\mu},\qquad \text{当 }0<r\leqslant 1\text{ 时}$$

$$\Pr(X\geqslant\mu r)\leqslant(r^{-r}e^{r-1})^{\mu},\qquad \text{当 }r\geqslant 1\text{ 时}$$

(b) 当 $r\approx 1$ 时，以方便的形式表达这些值的右边。(c) 证明如果 r 充分大，我们有 $\Pr(X\geqslant\mu r)\leqslant 2^{-\mu r}$。

23. [*HM23*] 试估计有由 $(q-pz)^{-n}$ 生成的负二项式分布的一个随机变量的尾概率，其中 $q=p+1$。

*1.2.11 渐近表示

我们通常需要知道一个近似的而不是精确的量，以便把它同另一个量作比较。例

如，当 n 很大时，对于 $n!$ 的斯特林近似值是这类型问题的一个很有用的代表。我们也利用了 $H_n \approx \ln n + \gamma$ 的事实。对于这样的渐近公式的推导一般都涉及高等数学，尽管在以下的几个小节中为得到我们所需要的结果，我们将使用的不过是初等微积分罢了。

***1.2.11.1 O 符号** Paul Bachmann 在*Analytische Zahlentheorie* (1894)中引进了对于近似的一个非常方便的符号。它就是符号 O，它允许我们以"="号来代替"≈"号并且量化精确的程度，例如

$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right) \tag{1}$$

（读作"H 下标 n，等于 n 的自然对数加上欧拉常数加上 n 分之一的大 O"。）

一般来说，每当 $f(n)$ 是正整数 n 的一个函数时，就可以用符号 $O(f(n))$。它代表一个不明确地知道的量，而只知道它的数值不会太大。$O(f(n))$ 的每一个出现便确切地意味着这样一点：有正常数 M 和 n_0 使得对于所有整数 $n \geqslant n_0$，由 $O(f(n))$ 表示的数 x_n 满足条件 $|x_n| \leqslant M|f(n)|$。我们不必说出常数 M 和 n_0 是什么，而且事实上这些常数对于 O 的每一个出现通常是不同的。

例如，等式(1)意味着当 $n \geqslant n_0$ 时，$|H_n - \ln n - \gamma| \leqslant M/n$。尽管常数 M 和 n_0 并未被说明，但我们可以确信，如果 n 足够地大，量 $O(1/n)$ 将是任意小的。

让我们多看些例子。已知

$$1^2 + 2^2 + \cdots + n^2 = \frac{1}{3}n(n+\frac{1}{2})(n+1) = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$$

所以由此得出

$$1^2 + 2^2 + \cdots + n^2 = O(n^4) \tag{2}$$

$$1^2 + 2^2 + \cdots + n^2 = O(n^3) \tag{3}$$

$$1^2 + 2^2 + \cdots + n^2 = \frac{1}{3}n^3 + O(n^2) \tag{4}$$

等式(2)是相当粗糙的，但并非不正确；等式(3)是一个更强的命题，而等式(4)还要强。为论证这些等式的正确性，我们必须证明，*如果 $P(n) = a_0 + a_1 n + \cdots + a_m n^m$ 是 m 次或更低次的任何多项式，则我们有 $P(n) = O(n^m)$*。这由下列推导得出，因为当 $n \geqslant 1$ 时，

$$|P(n)| \leqslant |a_0| + |a_1|n + \cdots + |a_m|n^m =$$

$$(|a_0|/n^m + |a_1|/n^{m-1} + \cdots + |a_m|)n^m \leqslant (|a_0| + |a_1| + \cdots + |a_m|)n^m$$

所以我们可以取 $M = |a_0| + |a_1| + \cdots + |a_m|$ 和 $n_0 = 1$。或者比如说可以取 $M = |a_0|/2^m + |a_1|/2^{m-1} + \cdots + |a_m|$ 和 $n_0 = 2$。

在近似的计算中，O 有很大的帮助，因为它简略地描述一个概念，这个概念经常出现，而它把通常无关紧要的那些细节信息掩盖掉。其次，可以按熟知的方式对它进行代数运算，尽管须记住它同代数的某些重要的差别。最重要的考虑是单向相等性的思想：

我们写$\frac{1}{2}n^2+n=O(n^2)$，但我们绝不写 $O(n^2)=\frac{1}{2}n^2+n$。（不然的话，由于$\frac{1}{4}n^2=O(n^2)$，我们就会引出荒谬的关系$\frac{1}{4}n^2=\frac{1}{2}n^2+n$。）我们总是使用下列的约定，即一个等式的右边不提供比其左边更多的信息，右边是左边的“粗略化”。

关于使用“=”号的约定可以更精确地表达如下：涉及 $O(f(n))$的公式可以看做是 n 的函数的集合。$O(f(n))$代表整数的所有这样的函数 g 的集合：即存在常数 M 和 n_0，使得对于所有整数 $n\geq n_0$，$|g(n)|\leq M|f(n)|$。如果 S 和 T 都代表函数的集合，则 $S+T$ 表示集合$\{g+h \mid g\in S$ 和 $h\in T\}$；我们以类似的方式定义 $S+c$，$S-T$，$S\cdot T$，$\log S$ 等等。同理，如果 $\alpha(n)$和 $\beta(n)$都是涉及 O 符号的公式，则记法 $\alpha(n)=\beta(n)$意味着由 $\alpha(n)$所表示的函数的集合被包含在以 $\beta(n)$所表示的集合中。

因此，对于“=”号可以实施我们已习惯去做的大多数运算，如果 $\alpha(n)=\beta(n)$，且 $\beta(n)=\gamma(n)$，则 $\alpha(n)=\gamma(n)$。而且，如果 $\alpha(n)=\beta(n)$，而如果 $\delta(n)$是在一个公式 $\gamma(n)$中以 $\beta(n)$来代替 $\alpha(n)$的某个出现所得到的一个公式，则 $\gamma(n)=\delta(n)$。这两个命题意味着，比如说，无论 $g(x_1,x_2,\cdots,x_n)$是什么样的实函数，而且如果对于 $1\leq k\leq m$，$\alpha_k(n)=\beta_k(n)$，则 $g(\alpha_1(n),\alpha_2(n),\cdots,\alpha_m(n))=g(\beta_1(n),\beta_2(n),\cdots,\beta_m(n))$。

下边是对于符号 O 可以做的某些简单的运算：

$$f(n)=O(f(n)) \tag{5}$$

$$c\cdot O(f(n))=O(f(n)),\quad \text{若 } c \text{ 是常数} \tag{6}$$

$$O(f(n))+O(f(n))=O(f(n)) \tag{7}$$

$$O(O(f(n)))=O(f(n)) \tag{8}$$

$$O(f(n))O(g(n))=O(f(n)g(n)) \tag{9}$$

$$O(f(n)g(n))=f(n)O(g(n)) \tag{10}$$

O 也经常用于$z=0$的邻域里复变量 z 的函数。我们写 $O(f(z))$来代表任何量 $g(z)$，使得每当$|z|<r$时就有$|g(z)|\leq M|f(z)|$。（和前边一样，M 和 r 是未确定的常数，尽管必要时我们可以确定之。）O 符号的上下文应当标识所涉及的变量以及该变量的作用域。当变量称做 n 时，我们默认 $O(f(n))$指的是一个很大整数 n 的函数；当变量叫做 z 时，我们默认 $O(f(z))$指的是一个小的复数 z 的函数。

假设 $g(z)$是由一个无穷级数

$$g(z)=\sum_{k\geq 0}a_kz^k$$

给出的函数，对于 $z=z_0$ 它收敛。则每当$|z|<|z_0|$时绝对值之和$\sum_{k\geq 0}|a_kz^k|$也收敛。因此如果 $z_0\neq 0$，我们总可以写

$$g(z)=a_0+a_1z+\cdots+a_mz^m+O(z^{m+1}) \tag{11}$$

因为我们有 $g(z)=a_0+a_1z+\cdots+a_mz^m+z^{m+1}(a_{m+1}+a_{m+2}z+\cdots)$；我们只须证明对于

某个正的 r，当 $|z|\leqslant r$ 时，带括号的量是有界的，因此每当 $|z|\leqslant r<|z_0|$ 时，容易看出 $|a_{m+1}|+|a_{m+2}|r+|a_{m+3}|r^2+\cdots$ 是一个上界。

例如在 1.2.9 小节列出的生成函数给了我们许多当 z 充分小时成立的重要的渐近公式，包括对于所有非负整数 m，

$$e^z = 1 + z + \frac{1}{2!}z^2 + \cdots + \frac{1}{m!}z^m + O(z^{m+1}) \tag{12}$$

$$\ln(1+z) = z - \frac{1}{2}z^2 + \cdots + \frac{(-1)^{m+1}}{m}z^m + O(z^{m+1}) \tag{13}$$

$$(1+z)^\alpha = 1 + \alpha z + \binom{\alpha}{2}z^2 + \cdots + \binom{\alpha}{m}z^m + O(z^{m+1}) \tag{14}$$

$$\frac{1}{1-z}\ln\frac{1}{1-z} = z + H_2z^2 + \cdots + H_mz^m + O(z^{m+1}) \tag{15}$$

重要的是注意到，由任何特定的 O 所隐含的被掩盖的常数 M 和 r 是彼此相关联的。例如，对于任何固定的 r，当 $|z|\leqslant r$ 时，函数 e^z 显然是 $O(1)$，因为 $|e^z|\leqslant e^{|z|}$；但对于 z 的所有值，没有常数 M 使得 $|e^z|\leqslant M$。因此我们需要随范围 r 的增大而使用越来越大的界 M。

有时一个渐近级数是正确的，尽管它并不对应于一个收敛的无穷级数。例如，借助于通常的幂表达阶乘幂的基本公式：

$$n^{\overline{r}} = \sum_{k=0}^{m}\left[{r \atop r-k}\right]n^{r-k} + O(n^{r-m-1}) \tag{16}$$

$$n^{\underline{r}} = \sum_{k=0}^{m}(-1)^k\left[{r \atop r-k}\right]n^{r-k} + O(n^{r-m-1}) \tag{17}$$

对于任何实数 r 和任何固定的整数 $m\geqslant 0$ 近似地成立。而对于所有的 n，和

$$\sum_{k=0}^{\infty}\left[{1/2 \atop 1/2-k}\right]n^{1/2-k}$$

发散（参见习题 12）。当然，当 r 是非负整数时，$n^{\overline{r}}$ 和 $n^{\underline{r}}$ 只不过是 r 次的多项式，而且(17)实质上和 1.2.6-(44)相同。当 r 是负整数且 $|n|>|r|$ 时，无穷和 $\sum_{k=0}^{\infty}\left[{r \atop r-k}\right]n^{r-k}$ 确实收敛到 $n^{\overline{r}}=1/(n-1)^{\underline{-r}}$；利用等式 1.2.6-(58)，这个和可以被写成更自然的形式 $\sum_{k=0}^{\infty}\left\{{k-r \atop -r}\right\}n^{r-k}$。

让我们对于迄今所介绍的概念给出一个简单的例子。考虑量 $\sqrt[n]{n}$；当 n 变成很大时，取 n 次方根的运算趋向于使值减小，但 $\sqrt[n]{n}$ 是减小还是增大并非立即就明显了的。结果是 $\sqrt[n]{n}$ 趋于 1。让我们考虑稍微更复杂的量 $n(\sqrt[n]{n}-1)$，现在当 n 变得更大时 $(\sqrt[n]{n}-1)$

变得更小;那 $n(\sqrt[n]{n}-1)$ 又将如何?

通过应用上边的公式可以很容易地求解这个问题,我们有

$$\sqrt[n]{n} = e^{\ln n/n} = 1 + (\ln n/n) + O((\ln n/n)^2) \tag{18}$$

因为当 $n \to \infty$ 时,$\ln n/n \to 0$,参见习题 8 和 11。这个等式证明了我们前边的论点即 $\sqrt[n]{n} \to 1$。其次,它告诉我们

$$n(\sqrt[n]{n}-1) = n(\ln n/n + O((\ln n/n)^2)) = \ln n + O((\ln n)^2/n) \tag{19}$$

换言之,$n(\sqrt[n]{n}-1)$ 近似地等于 $\ln n$;其差为 $O((\ln n)^2/n)$,当 n 趋于无穷大时它趋于零。

人们常常通过假定 O 给出精确的增长的阶,滥用这个符号;他们使用它就如同它确定一个下界,也确定一个上界似的。例如,对于 n 个数进行排序的一个算法可以说是低效的,"因为它的运行时间是 $O(n^2)$",然而 $O(n^2)$ 的运行时间未必意味着这个运行时间不是 $O(n)$ 的。还有另外一个符号 Ω,表示下界:表述

$$g(n) = \Omega(f(n)) \tag{20}$$

意思是存在正常数 L 和 n_0 使得对所有 $n \geqslant n_0$,

$$|g(n)| \geqslant L|f(n)|$$

利用这个符号我们可以正确地下结论说,其运行时间为 $\Omega(n^2)$ 的一个排序算法将不如其运行时间为 $O(n \log n)$ 的算法那样有效,如果 n 足够大的话。然而,如果不知道由 O 和 Ω 所隐含的常数因子,对于从多大的 n 开始 $O(n \log n)$ 方法才开始胜出我们却无可奉告。

最后,如果我们要指出增长的精确的阶而不涉及精确的常数因子,我们可以使用 Θ 符号:

$$g(n) = \Theta(f(n)) \Leftrightarrow g(n) = O(f(n)) \text{ 和 } g(n) = \Omega(f(n)) \tag{21}$$

习　题

1. [*HM01*] $\lim\limits_{n\to\infty} O(n^{-1/3})$ 等于什么?

▶ 2. [*M10*] 通过使用"自显然"的公式 $O(f(n)) - O(f(n)) = 0$,B.C.Dull 先生得出了令人惊讶的结果,他的错误是什么,他的公式右边应当是什么?

3. [*M15*] 试把 $(\ln n + \gamma + O(1/n))$ 乘以 $(n + O(\sqrt{n}))$,并以 O 符号表达你的答案。

▶ 4. [*M15*] 如果 $a > 0$,试给出 $n(\sqrt[n]{a}-1)$ 的一个渐近展开到 $O(1/n^3)$ 项。

5. [*M20*] 证明或者反驳:如果对于所有的 n,$f(n)$ 和 $g(n)$ 都为正,则 $O(f(n)+g(n)) = f(n) + O(g(n))$。(试同(10)作比较。)

▶ 6. [*M20*] 下列论证的错误何在?"由于 $n = O(n)$,且 $2n = O(n)$,…,我们有

$$\sum_{k=1}^{n} kn = \sum_{k=1}^{n} O(n) = O(n^2)\text{"}$$

7. [*HM15*] 证明:如果 m 是任意整数,则对于任意大的 x 值,不存在 M 使得 $e^x \leqslant Mx^m$。

8. [*HM20*] 证明：当 $n\to\infty$ 时，$(\ln n)^m/n\to 0$。

9. [*HM20*] 证明对于所有固定的 $m\geqslant 0$，$e^{O(z^m)}=1+O(z^m)$。

10. [*HM22*] 试对 $\ln(1+O(z^m))$ 作出类似于习题 9 那样的论断。

▶ 11. [*M11*] 试说明为什么等式(18)为真。

12. [*HM25*] 利用 $\left[{1/2 \atop 1/2-k}\right]=(-\frac{1}{2})^k[z^k](ze^2/(e^2-1))^{1/2}$ 这一事实，证明对于任何整数 n，当 $k\to\infty$ 时，$\left[{1/2 \atop 1/2-k}\right]n^{-k}$ 不趋于零。

▶ 13. [*M10*] 证明或反驳：当且仅当 $f(n)=O(g(n))$ 时，$g(n)=\Omega(f(n))$。

* **1.2.11.2 欧拉求和公式** 为得到对于一个和的好的近似，最有用的方法之一是由欧拉给出的。他的方法通过一个积分来近似一个有限和，而且在许多情况下它给了我们获得越来越好的近似的一个手段。[*Commentarii Academiæ Scientiarum Petropolitanæ* **6** (1732), 68~97。]

图 12 显示了当 $n=7$ 时，对 $\int_1^n f(x)\mathrm{d}x$ 和 $\sum_{k=1}^{n-1}f(k)$ 的比较。假定 $f(x)$ 是一个可微分的函数，欧拉的策略导致了对于这两个量的差的一个有用的公式。

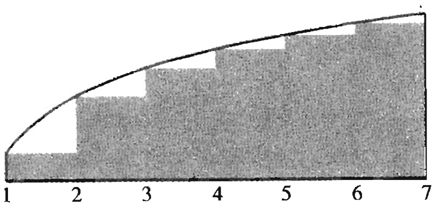


图 12 一个和数与一个积分之比较

为方便起见，我们将使用记号

$$\{x\}=x \bmod 1=x-\lfloor x\rfloor \tag{1}$$

我们的推导以下列的恒等式开始

$$\int_k^{k+1}(\{x\}-\frac{1}{2})f'(x)\mathrm{d}x=(x-k-\frac{1}{2})f(x)\Big|_k^{k+1}-\int_k^{k+1}f(x)\mathrm{d}x=\frac{1}{2}(f(k+1)+f(k))-\int_k^{k+1}f(x)\mathrm{d}x \tag{2}$$

（这由分部积分得出。）对于 $1\leqslant k<n$，对这个等式的两边作加法，我们求得

$$\int_1^n(\{x\}-\frac{1}{2})f'(x)\mathrm{d}x=\sum_{1\leqslant k<n}f(k)+\frac{1}{2}(f(n)-f(1))-\int_1^n f(x)\mathrm{d}x$$

即是

$$\sum_{1\leqslant k<n} f(k)=\int_1^n f(x)\mathrm{d}x-\frac{1}{2}(f(n)-f(1))+\int_1^n B_1(\{x\})f'(x)\mathrm{d}x \tag{3}$$

其中 $B_1(x)$是多项式 $x-\frac{1}{2}$。这就是和与积分之间所求的关系式。

如果我们继续进行分部积分，则这个近似化还可进一步进行下去。然而在这样做之前，我们需要讨论一下伯努利数，它们是下列无穷级数的系数

$$\frac{z}{\mathrm{e}^z-1}=B_0+B_1z+\frac{B_2z^2}{2!}+\cdots=\sum_{k\geqslant 0}\frac{B_kz^k}{k!} \tag{4}$$

这个级数的系数出现在广泛的问题中，它是由 James Bernoulli（伯努利）在他的*Ars Conjectandi* 中引入的，于 1713 年在伯努利死后才发表。巧的是，大约在同一时期，它们也由日本的关孝和（Takakazu Seki）所发现，也是在他死后不久于 1712 年才首先发表出来。［参见关孝和的*Collected Works*（*Osaka*：1974），39～42。］

我们有

$$B_0=1,\quad B_1=-\frac{1}{2},\quad B_2=\frac{1}{6},\quad B_3=0,\quad B_4=-\frac{1}{30} \tag{5}$$

附录 A 中给出更多的值。由于

$$\frac{z}{\mathrm{e}^z-1}+\frac{z}{2}=\frac{z}{2}\frac{\mathrm{e}^z+1}{\mathrm{e}^z-1}=-\frac{z}{2}\frac{\mathrm{e}^{-z}+1}{\mathrm{e}^{-z}-1}$$

是一个偶函数，我们看到

$$B_3=B_5=B_7=B_9=\cdots=0 \tag{6}$$

如果我们以 e^2-1 来乘定义的等式(4)的两边并等置 z 的相同次幂的系数，我们便得到公式

$$\sum_k\binom{n}{k}B_k=B_n+\delta_{n1} \tag{7}$$

（参见习题 1。）我们现在定义伯努利多项式

$$B_m(x)=\sum_k\binom{m}{k}B_kx^{m-k} \tag{8}$$

如果 $m=1$，则 $B_1(x)=B_0x+B_1=x-\frac{1}{2}$，对应于上边等式(3)中所用的多项式。如果 $m>1$，由(7)我们有 $B_m(1)=B_m=B_m(0)$；换言之，$B_m(\{x\})$在整数点 x 处不连续。

不久就会清楚伯努利多项式和伯努利数对于我们的问题的关联，通过对等式(8)求微分得出

$$B'_m(x) = \sum_k \binom{m}{k}(m-k)B_k x^{m-k-1} =$$

$$m\sum_k \binom{m-1}{k} B_k x^{m-1-k} =$$

$$mB_{m-1}(x) \tag{9}$$

因此当 $m\geqslant 1$ 时,我们可进行分部积分如下:

$$\frac{1}{m!}\int_1^n B_m(\{x\})f^{(m)}(x)\mathrm{d}x = \frac{1}{(m+1)!}(B_{m+1}(1)f^{(m)}(n) - B_{m+1}(0)f^{(m)}(1)) - \frac{1}{(m+1)!}\int_1^n B_{m+1}(\{x\})f^{(m+1)}(x)\mathrm{d}x$$

由这一结果我们可以继续改进近似式等式(3),而且利用(6),得到欧拉的一般公式

$$\sum_{1\leqslant k<n} f(k) = \int_1^n f(x)\mathrm{d}x - \frac{1}{2}(f(n) - f(1)) + \frac{B_2}{2!}(f'(n) - f'(1)) + \cdots + \frac{(-1)^m B_m}{m!}(f^{(m-1)}(n) - f^{(m-1)}(1)) + R_{mn} =$$

$$\int_1^n f(x)\mathrm{d}x + \sum_{k=1}^m \frac{B_k}{k!}(f^{(k-1)}(n) - f^{(k-1)}(1)) + R_{mn} \tag{10}$$

其中

$$R_{mn} = \frac{(-1)^{m+1}}{m!}\int_1^n B_m(\{x\})f^{(m)}(x)\mathrm{d}x \tag{11}$$

当 $B_m(\{x\})f^{(m)}(x)/m!$ 非常小时,余式 R_{mn} 也将很小,而且事实上,当 m 是偶数时可以证明

$$\left|\frac{B_m(\{x\})}{m!}\right| \leqslant \frac{|B_m|}{m!} < \frac{4}{(2\pi)^m} \tag{12}$$

[参见*CMath*, §9.5。]另一方面,通常的结果是 $f^{(m)}(x)$ 的数值随 m 的增加而变大,因此当给定 n 时有一个令 $|R_{mn}|$ 具有最小值的“最好”的值 m。

已知当 m 为偶数时,只要对于 $1<x<n$, $f^{(m+2)}(x)f^{(m+4)}(x)>0$,有一个数 θ 使得

$$R_{mn} = \theta\frac{B_{m+2}}{(m+2)!}(f^{(m+1)}(n) - f^{(m+1)}(1)),\quad 0<\theta<1 \tag{13}$$

所以在这些情况下,余式同头一个抛弃的项同号,而且小于这个被抛弃的项。习题 3 中证明了这个结果的较简单的形式。

现在让我们应用欧拉公式到某些重要的例子上。首先置 $f(x)=1/x$，导数是 $f^{(m)}(x)=(-1)^m m!/x^{m+1}$，所以根据等式(10)有

$$H_{n-1} = \ln n + \sum_{k=1}^{m} \frac{B_k}{k}(-1)^{k-1}\left(\frac{1}{n^k}-1\right) + R_{mn} \tag{14}$$

现在我们求得

$$\gamma = \lim_{n\to\infty}(H_{n-1}-\ln n) = \sum_{k=1}^{m}\frac{B_k}{k}(-1)^k + \lim_{n\to\infty} R_{mn} \tag{15}$$

$\lim\limits_{n\to\infty} R_{mn} = -\int_1^\infty B_m(\{x\})dx/x^{m+1}$ 存在 这一事实证明常数 γ 确实存在。因此可以把等式(14)和(15)合在一起，来推导出对于调和数的一个一般的近似：

$$H_{n-1} = \ln n + \gamma + \sum_{k=1}^{m}\frac{(-1)^{k-1}B_k}{kn^k} + \int_n^\infty \frac{B_m(\{x\})dx}{x^{m+1}} =$$

$$\ln n + \gamma + \sum_{k=1}^{m-1}\frac{(-1)^{k-1}B_k}{kn^k} + O\left(\frac{1}{n^m}\right)$$

以 $m+1$ 代替 m，得

$$H_{n-1} = \ln n + \gamma + \sum_{k=1}^{m}\frac{(-1)^{k-1}B_k}{kn^k} + O\left(\frac{1}{n^{m+1}}\right) \tag{16}$$

其次，由等式(13)可看出误差小于被抛弃的头一项。作为一个特殊情况我们有(两边都加上 $1/n$)

$$H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \epsilon,\quad 0<\epsilon<\frac{B_6}{6n^6} = \frac{1}{252n^6}$$

这即是等式 1.2.7-(3)，对于大的 k 值伯努利数变得非常大(当 n 为偶数时近似于 $(-1)^{1+k/2}2(k!/(2\pi)^k)$)，因此对于任何固定的 n 的值，等式(16)不可能被推广成一个收敛的无穷级数。

同样的技术可以被用来推导斯特林的近似。这次我们置 $f(x)=\ln x$，于是等式(10)产生出

$$\ln(n-1)! = n\ln n - n + 1 - \frac{1}{2}\ln n + \sum_{1<k\leq m}\frac{B_k(-1)^k}{k(k-1)}\left(\frac{1}{n^{k-1}}-1\right) + R_{mn} \tag{17}$$

如同上边那样进行，我们发现极限

$$\lim_{n\to\infty}\left(\ln n! - n\ln n + n - \frac{1}{2}\ln n\right) = 1 + \sum_{1<k\leq m}\frac{B_k(-1)^{k+1}}{k(k-1)} + \lim_{n\to\infty}R_{mn}$$

存在；暂时把它叫做 σ(斯特林常数)。于是得到斯特林的结果

$$\ln n! = (n+\frac{1}{2})\ln n - n + \sigma + \sum_{1<k\leq m}\frac{B_k(-1)^k}{k(k-1)n^{k-1}} + O\left(\frac{1}{n^m}\right) \quad (18)$$

特别地，令 $m=5$，我们有

$$\ln n! = (n+\frac{1}{2})\ln n - n + \sigma + \frac{1}{12n} - \frac{1}{360n^3} + O\left(\frac{1}{n^5}\right)$$

现在可以对两边取乘幂

$$n! = e^{\sigma}\sqrt{n}\left(\frac{n}{e}\right)^n \exp(\frac{1}{12n} - \frac{1}{360n^3} + O(\frac{1}{n^5}))$$

利用 $e^{\sigma}=\sqrt{2\pi}$ 的事实(参见习题 5)，并展开这个乘幂，便得到最后的结果

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \frac{1}{12n} + \frac{1}{288n^2} - \frac{139}{51840n^3} - \frac{571}{2488320n^4} + O\left(\frac{1}{n^5}\right)\right) \quad (19)$$

习 题

1. [*M18*] 证明等式(7)。

2. [*HM20*] 注意对于任意序列 B_n，而不仅仅对于由等式(4)定义的序列，等式(9)由等式(8)得出。说明为什么对于等式(10)的正确性说来后一序列是必要的。

3. [*HM20*] 令 $C_{mn}=(B_m/m!)(f^{(m-1)}(n)-f^{(m-1)}(1))$是在欧拉的求和公式中第 m 个正确的项。假定 $f^{(m)}(x)$对于 $1\leq x\leq n$ 有一固定的符号；证明当 $m=2k>0$ 时，$|R_{mn}|\leq|C_{mn}|$。换言之，证明余式的绝对值不大于所计算的最后项。

▶ 4. [*HM20*](乘幂之和) 当 $f(x)=x^m$ 时，f 的高阶导数全为 0，所以借助于伯努利数，欧拉求和公式给出对于和数

$$S_m(n) = \sum_{0\leq k<n} k^m$$

的一个精确值。(正是对于 $m=1,2,3,\cdots$，对 $S_m(n)$的研究导致伯努利和关孝和首先发现那些数。)试借助于伯努利多项式来表达 $S_m(n)$。对于 $m=0,1$ 和 2，检查你的答案。(注意，对于$0\leq k<n$ 而不是 $1\leq k<n$ 来实现所求的和；欧拉公式自始至终地可以以 0 代替 1 来应用。)

5. [*HM30*] 给定

$$n! = \kappa\sqrt{n}(\frac{n}{e})^n(1+O(\frac{1}{n}))$$

通过 Wallis 乘积(习题 1.2.5-18)证明 $\kappa=\sqrt{2\pi}$。[提示：对于 n 的很大的值考虑$\binom{2n}{n}$。]

▶ 6. [*HM30*] 证明斯特林近似对于非整数的 n 也成立：

$$\Gamma(x+1) = \sqrt{2\pi x}(\frac{x}{e})^x(1+O(\frac{1}{x})), \qquad x\geq a>0$$

[提示：在欧拉求和公式中令 $f(x)=\ln(x+c)$，并应用在 1.2.5 小节中给出的 $\Gamma(x)$的定义。]

▶ 7. [*HM32*] $1^1 2^2 3^3\cdots n^n$的近似值是什么？

8. [*M23*] 试求 $\ln(an^2+bn)!$ 带有绝对误差 $O(n^{-2})$的近似值，使用它计算当 c 是一个正

常数时$\binom{cn^2}{n}\Big/c^n\binom{n^2}{n}$带有相对误差$O(n^{-2})$的近似值。这里绝对误差$\epsilon$指(真值)=(近似值)$+\epsilon$;相对误差$\epsilon$指(真值)=(近似值)$(1+\epsilon)$。

▶ 9. [*M25*] 以两种方法求$\binom{2n}{n}$带有$O(n^{-3})$的相对误差的近似值:(a)通过斯特林近似;(b)通过习题 1.2.6-47 和式 1.2.11.1-(16)。

***1.2.11.3　一些近似计算**　在本小节中我们将研究以下三个令人感兴趣的和式,为的是导出它们的近似值:

$$P(n)=1+\frac{n-1}{n}+\frac{n-2}{n}\frac{n-2}{n-1}+\cdots=\sum_{k=0}^{n}\frac{(n-k)^k(n-k)!}{n!}\tag{1}$$

$$Q(n)=1+\frac{n-1}{n}+\frac{n-1}{n}\frac{n-2}{n}+\cdots=\sum_{k=1}^{n}\frac{n!}{(n-k)!\,n^k}\tag{2}$$

$$R(n)=1+\frac{n}{n+1}+\frac{n}{n+1}\frac{n}{n+2}+\cdots=\sum_{k\geqslant 0}\frac{n!\;n^k}{(n+k)!}\tag{3}$$

这些函数在表面上很类似,但是实质上却很不同。它们出现在我们稍后将要遇到的若干算法中。$P(n)$和$Q(n)$两者是有限和,而$R(n)$是一个无穷和。看起来当n很大时,所有这三个和将近乎相等,尽管任何一个的近似值将是什么并不明显。对于这些函数的近似值问题的研究将为我们导出一些非常有启示的附带的结果。(在往下进行看看这些问题是如何被解决之前,你可能愿意暂时停下来,而亲自尝试研究这些问题的求解。)

首先观察$Q(n)$和$R(n)$之间的一个重要联系:

$$Q(n)+R(n)=\frac{n!}{n^n}\Big(\Big(1+n+\cdots+\frac{n^{n-1}}{(n-1)!}\Big)+\Big(\frac{n^n}{n!}+\frac{n^{n+1}}{(n+1)!}+\cdots\Big)\Big)=$$

$$\frac{n!\;e^n}{n^n}\tag{4}$$

斯特林公式告诉我们,$n!\,e^n/n^n$近似于$\sqrt{2\pi n}$,所以我们猜测$Q(n)$和$R(n)$每一个将大致等于$\sqrt{\pi n/2}$。

为往下进行,必须考虑对于e^n的级数的部分和,通过利用带余式的泰勒公式

$$f(x)=f(0)+f'(0)x+\cdots+\frac{f^{(n)}(0)x^n}{n!}+\int_0^x\frac{t^n}{n!}f^{(n+1)}(x-t)\,dt\tag{5}$$

我们立即导出一个重要的函数,通常称为不完全伽玛函数:

$$\gamma(a,x)=\int_0^x e^{-t}t^{a-1}dt \tag{6}$$

我们将假定 $a<0$。由习题 1.2.5-20,我们有 $\gamma(a,\infty)=\Gamma(a)$;这就是取名作“不完全伽玛函数”的理由。它有两个有用的关于 x 的乘幂的级数展开(参见习题 2 和 3):

$$\gamma(a,x)=\frac{x^a}{a}-\frac{x^{a+1}}{a+1}+\frac{x^{a+2}}{2!(a+2)}-\cdots=\sum_{k\geq 0}\frac{(-1)^k x^{k+a}}{k!(k+a)} \tag{7}$$

$$e^x\gamma(a,x)=\frac{x^a}{a}+\frac{x^{a+1}}{a(a+1)}+\frac{x^{a+2}}{a(a+1)(a+2)}+\cdots=\sum_{k\geq 0}\frac{x^{k+a}}{a(a+1)\cdots(a+k)} \tag{8}$$

由第二个公式,我们看出与 $R(n)$的联系:

$$R(n)=\frac{n!\ e^n}{n^n}\left(\frac{\gamma(n,n)}{(n-1)!}\right) \tag{9}$$

这个等式被有意地写成比它应有的要更为复杂的形式,因为 $\gamma(n,n)$是 $\gamma(n,\infty)=\Gamma(n)=(n-1)!$的一部分,而 $n!\ e^n/n^n$ 是(4)中的量。

问题归结成得出 $\gamma(n,n)/(n-1)!$ 的好的估计。我们现在将确定当 y 固定和 x 很大时 $\gamma(x+1,x+y)/\Gamma(x+1)$的近似值。这里要采用的方法比结果还重要,因此读者应仔细地研究以下的推导。

由定义,我们有

$$\frac{\gamma(x+1,x+y)}{\Gamma(x+1)}=\frac{1}{\Gamma(x+1)}\int_0^{x+y}e^{-t}t^x dt=$$

$$1-\frac{1}{\Gamma(x+1)}\int_x^{\infty}e^{-t}t^x dt+\frac{1}{\Gamma(x+1)}\int_x^{x+y}e^{-t}t^x dt \tag{10}$$

让我们置

$$I_1=\int_x^{\infty}e^{-t}t^x dt$$

$$I_2=\int_x^{x+y}e^{-t}t^x dt$$

并且依次考虑每个积分。

对于 I_1 的估计:通过替换 $t=x(1+u)$,我们把 I_1 转换成从 0 到无穷的积分;进一步替换 $v=u-\ln(1+u)$, $dv=(1-1/(1+u))du$,这样做是合法的,因为 v 是 u 的一个单调函数:

$$I_1 = e^{-x}x^x\int_0^\infty xe^{-xu}(1+u)^x du = e^{-x}x^x\int_0^\infty xe^{-xv}(1+\frac{1}{u})dv \tag{11}$$

在最后的积分中我们将以 v 的一个幂级数来代替 $1+1/u$。我们有

$$v = \frac{1}{2}u^2 - \frac{1}{3}u^3 + \frac{1}{4}u^4 - \frac{1}{5}u^5 + \cdots = (u^2/2)(1 - \frac{2}{3}u + \frac{1}{2}u^2 - \frac{2}{5}u^3 + \cdots)$$

置 $w = \sqrt{2v}$,因此我们有

$$w = u(1 - \frac{2}{3}u + \frac{1}{2}u^2 - \frac{2}{5}u^3 + \cdots)^{1/2} =$$
$$u - \frac{1}{3}u^2 + \frac{7}{36}u^3 - \frac{73}{540}u^4 + \frac{1331}{12960}u^5 + O(u^6)$$

(这个展开可以通过二项式定理得到,在 4.7 节中会讨论实现这样的变换的有效方法,以及实现以下所需要的其它幂级数操作的有效方法。)现在我们可以把 u 作为关于 w 的幂级数来求解:

$$u = w + \frac{1}{3}w^2 + \frac{1}{36}w^3 - \frac{1}{270}w^4 + \frac{1}{4320}w^5 + O(w^6)$$

$$1 + \frac{1}{u} = 1 + \frac{1}{w} - \frac{1}{3} + \frac{1}{12}w - \frac{2}{135}w^2 + \frac{1}{864}w^3 + O(w^4) = \tag{12}$$
$$\frac{1}{\sqrt{2}}v^{-1/2} + \frac{2}{3} + \frac{\sqrt{2}}{12}v^{1/2} - \frac{4}{135}v + \frac{\sqrt{2}}{432}v^{3/2} + O(v^2)$$

在所有这些公式中,O 符号指的都是小的自变量值,即是对于充分小的正数 r,$|u| \leqslant r$, $|v| \leqslant r$, $|w| \leqslant r$。这是足够好的吗?对于 $0 \leqslant v < \infty$,而不仅仅是对于 $|v| \leqslant r$,在等式(11)中借助于 v 对 $1+1/u$ 的替换应当是有效的。幸而,结果是,从 0 到 ∞ 的积分的值几乎完全依赖于接近零处的被积函数的值。事实上,对于任何固定的 $r>0$ 及对于很大的 x,我们有(参见习题 4)

$$\int_r^\infty xe^{-xv}(1+\frac{1}{u})dv = O(e^{-rx}) \tag{13}$$

我们对直到项 $O(x^{-m})$ 为止的一个近似感兴趣,因为对于任何正 r 和 m, $O((1/e^r)^x)$ 比 $O(x^{-m})$ 要小得多,对于任何固定的正的 r,仅需要从 O 到 r 进行积分。因此我们取 r 足够小,使得上边所完成的所有幂级数操作都是完全正确的(参见等式 1.2.11.1-(11)和 1.2.11.3-(13))。

现在

$$\int_0^\infty xe^{-xv}v^\alpha dv = \frac{1}{x^\alpha}\int_0^\infty e^{-q}q^\alpha dq = \frac{1}{x^\alpha}\Gamma(\alpha+1), \text{若 } \alpha > -1 \tag{14}$$

因此通过把级数(12)插入积分(11)当中,最后得

$$I_1 = e^{-x}x^x(\sqrt{\frac{\pi}{2}}\ x^{1/2} + \frac{2}{3} + \frac{\sqrt{2\pi}}{24}x^{-1/2} - \frac{4}{135}x^{-1} + \frac{\sqrt{2\pi}}{576}x^{-3/2} + O(x^{-2})) \tag{15}$$

对 I_2 的估计:在积分 I_2 中,替换 $t = u + x$ 并得到

$$I_2 = e^{-x}x^x\int_0^y e^{-u}(1 + \frac{u}{x})^x du \tag{16}$$

现在对于 $0 \leqslant u \leqslant y$ 和大的 x,

$$e^{-u} = (1 + \frac{u}{x})^x = \exp(-u + x\ln(1 + \frac{u}{x})) = \exp(\frac{-u^2}{2x} + \frac{u^3}{3x^2} + O(x^{-3})) =$$

$$1 - \frac{u^2}{2x} + \frac{u^4}{8x^2} + \frac{u^3}{3x^3} + O(x^{-3})$$

因此求得

$$I_2 = e^{-x}x^x(y - \frac{y^3}{6}x^{-1} + (\frac{y^4}{12} + \frac{y^5}{40})x^{-2} + O(x^{-3})) \tag{17}$$

最后,我们分析当以(10)中的因式 $1/\Gamma(x+1)$ 来乘等式(15)和(17)时出现的系数 $e^{-x}x^x/\Gamma(x+1)$,由斯特林近似(由习题 1.2.11.2-6 它对于伽玛函数是正确的),我们有

$$\frac{e^{-x}x^x}{\Gamma(x+1)} = \frac{e^{-1/12x + O(x^{-3})}}{\sqrt{2\pi x}} =$$

$$\frac{1}{\sqrt{2\pi}}x^{-1/2} - \frac{1}{12\sqrt{2\pi}}x^{-3/2} + \frac{1}{288\sqrt{2\pi}}x^{-5/2} + O(x^{-7/2}) \tag{18}$$

现在来个大总结:等式(10),(15),(17)和(18)在一起产生

定理 A *对于很大的 x 值和固定的 y,*

$$\frac{\gamma(x+1, x+y)}{\Gamma(x+1)} = \frac{1}{2} + (\frac{y - 2/3}{\sqrt{2\pi}})x^{-1/2} + \frac{1}{\sqrt{2\pi}}(\frac{23}{270} - \frac{y}{12} - \frac{y^3}{6})x^{-3/2} + O(x^{-5/2}) \blacksquare \tag{19}$$

我们所使用的方法表明可以如何随心所欲地把这个近似推广到 x 的幂级数。

通过使用等式(4)和(9),定理 A 可用来得到 $R(n)$ 和 $Q(n)$ 的近似值,但我们将把这个计算推迟到稍后的某个时候。现在让我们转到 $P(n)$,对于它似乎要求稍微不同的方法,我们有

$$P(n) = \sum_{k=0}^{n}\frac{k^{n-k}k!}{n!} = \frac{\sqrt{2\pi}}{n!}\sum_{k=0}^{n}k^{n+1/2}e^{-k}(1 + \frac{1}{12k} + O(k^{-2})) \tag{20}$$

因此为了得到 $P(n)$ 的值,必须研究形如

$$\sum_{k=0}^{n}k^{n+1/2}e^{-k}$$

的和。

令 $f(x) = x^{n+1/2}e^{-x}$ 并应用欧拉求和公式

$$\sum_{k=0}^{n}k^{n+1/2}e^{-k} = \int_0^n x^{n+1/2}e^{-x}dx + \frac{1}{2}n^{n+1/2}e^{-n} + \frac{1}{24}n^{n-1/2}e^{-n} - R \tag{21}$$

对余式(参见习题 5)的一个粗略分析表明 $R=O(n^n e^{-n})$;而且由于积分是一个不完全的伽玛函数,我们有

$$\sum_{k=0}^{n} k^{n+1/2}e^{-k}=\gamma(n+\frac{3}{2},n)+\frac{1}{2}n^{n+1/2}e^{-n}+O(n^n e^{-n}) \qquad (22)$$

公式(20)也要求对下列的和

$$\sum_{k=0}^{n} k^{n-1/2}e^{-k}=\sum_{0\le k\le n-1} k^{(n-1)+1/2}e^{-k}+n^{n-1/2}e^{-n}$$

的估计,这也可以通过等式(22)得到。

现在在我们手头已有足够的公式可用以确定 $P(n)$,$Q(n)$和 $R(n)$的近似值,它不过是替换,作乘法等而已。在这一过程中,我们有时还要用到展开式

$$(n+\alpha)^{n+\beta}=n^{n+\beta}e^{\alpha}(1+\alpha(\beta-\frac{\alpha}{2})\frac{1}{n}+O(n^{-2})) \qquad (23)$$

其证明在习题 6 给出。(21)的方法仅生成 $P(n)$的渐近级数的头两项;通过使用习题 14 中描述的具有启发性的技术可以得到进一步的项。

所有这些计算的结果给了我们所求的渐近公式

$$P(n)=\sqrt{\frac{\pi n}{2}}-\frac{2}{3}+\frac{11}{24}\sqrt{\frac{\pi}{2n}}+\frac{4}{135n}-\frac{71}{1152}\sqrt{\frac{\pi}{2n^3}}+O(n^{-2}) \qquad (24)$$

$$Q(n)=\sqrt{\frac{\pi n}{2}}-\frac{1}{3}+\frac{1}{12}\sqrt{\frac{\pi}{2n}}-\frac{4}{135n}+\frac{1}{288}\sqrt{\frac{\pi}{2n^3}}+O(n^{-2}) \qquad (25)$$

$$R(n)=\sqrt{\frac{\pi n}{2}}+\frac{1}{3}+\frac{1}{12}\sqrt{\frac{\pi}{2n}}+\frac{4}{135n}+\frac{1}{288}\sqrt{\frac{\pi}{2n^3}}+O(n^{-2}) \qquad (26)$$

这里所研究的函数在公开发表的著作中都只是略做讨论。$P(n)$的展开式中的头一项$\sqrt{\pi n/2}$是由 H.B.Demuth 给出的[斯坦福大学博士论文(1956 年 10 月),67 ~ 68]。利用这一结果,对于 $n\le 2000$ 的 $P(n)$的数值表,以及一个好的计算尺,作者于 1963 年进行了经验估计 $P(n)\approx\sqrt{\pi n/2}-0.6667+0.575/\sqrt{n}$。自然猜测 0.6667 实际上是 2/3 的一个近似,而 0.575 将或许就是 $\gamma=0.57721\cdots$的一个近似(为什么不乐观呢?)。稍后在本节被写成时,才建立了 $P(n)$正确的展开,而猜测 2/3被证实了。至于另一个系数 0.575 不是 γ 而是$\frac{11}{24}\sqrt{\frac{\pi}{2}}\approx 0.5744$。这就漂亮地既证实了理论又证实了经验估计。

等价于 $Q(n)$和 $R(n)$的渐近值的公式首先由卓越的印度自学数学家 S. Ramanujan 确定的。他在*J. Indian Math. Soc.* **3** (1911),128; **4** (1912),151 ~ 152 上提出了估计$n!\,e^n/2n^n-Q(n)$的问题。在对于这个问题的回答中,他给出的渐近级数$\frac{1}{3}+\frac{4}{135}n^{-1}-\frac{8}{2835}n^{-2}-\frac{16}{8505}n^{-3}+\cdots$,相当大地超过等式(25)。他的推导比起上边叙述的

方法显得更优美；为了估计出 I_1，他替换 $t = x + u\sqrt{2x}$，并把被积函数表达为形如 $c_{jk}\int_0^{\infty} \exp(-u^2) u^j x^{-k/2} du$ 的项之和。积分 I_2 完全可以避免，因为当 $a > 0$ 时，$a\gamma(a, x) = x^a e^{-x} + \gamma(a+1, x)$；参见(8)。习题 20 中出现有求 $Q(n)$ 的近似的一个甚至更简单的方法，它或许是最简单的了。我们所用的推导，尽管不必要地复杂化，但都是有启发性的。它是由 R. Furch 给出的[*Zeitschrift für Physik* **112** (1939)，92 ~ 95]，他主要是对使得 $\gamma(x+1, x+y) = \Gamma(x+1)/2$ 的 y 值感兴趣。不完全的伽玛函数的渐近性质后来被 F.G. Tricomi 推广到复变量的情形[*Math. Zeitschrift* **53** (1950)，136 ~ 148]。也请参见 N.M. Temme，*Math. Comp.* **29** (1975)，1109 ~ 1114；*SIAM J. Math. Anal.* **10** (1979)，757 ~ 766。在 *AMM* **75** (1968)，1019 ~ 1021 上，H.W.Gould 列出了对 $Q(n)$ 的若干其它研究的参考文献。

对于 $P(n)$，$Q(n)$ 和 $R(n)$ 的渐近级数的推导仅仅使用初等微积分的简单技术。注意，对于每一个函数我们已经使用不同的方法！实际上，完全可以使用习题 14 的技术来解所有三个问题，在 5.1.4 和 5.2.2 小节要对这些技术做进一步的说明。那将是更为优雅的，但就稍逊启发性了。

关于进一步的介绍，有兴趣的读者可查阅 N.G.de Bruijn 的佳作*Asymptotic Methods in Analysis* (Amsterdam: North-Holland, 1961)。也可参见由 A.M.Odlyzko 所写的更新的评述[*Handbook of Combinatorics* **2** (MIT Press, 1995)，1063 ~ 1229]，它包含了 65 个详细的例子和大量的参考文献。

习　题

1.[*HM20*] 由对 n 的归纳法证明等式(5)。

2.[*HM20*] 试由等式(6)求得等式(7)。

3.[*M20*] 试由等式(7)推导出等式(8)。

▶ 4.[*HM10*] 证明等式(13)。

5.[*HM24*] 试证明等式(21)中的 R 是 $O(n^n e^{-n})$。

▶ 6.[*HM20*] 证明等式(23)。

▶ 7.[*HM30*] 在计算 I_2 时，我们必须考虑

$$\int_0^y e^{-u}\left(1 + \frac{u}{x}\right)^x du$$

当 y 是固定的和 x 很大时，试给出

$$\int_0^{yx^{1/4}} e^{-u}\left(1 + \frac{u}{x}\right)^x du$$

到阶 $O(x^{-2})$ 的项的渐近表达。

8.[*HM30*] 如果当 $x \to \infty$ 和 $0 \le r < 1$ 时 $f(x) = O(x^r)$，证明，如果 $m = \lceil (s+2r)/(1-r) \rceil$，则

$$\int_0^{f(x)} e^{-u}\left(1 + \frac{u}{x}\right)^x du = \int_0^{f(x)} \exp\left(\frac{-u^2}{2x} + \frac{u^3}{3x^2} - \cdots + \frac{(-1)^{m-1}u^m}{mx^{m-1}}\right) du + O(x^{-s})$$

[这特别地证明了由 Tricomi 给出的一个结果：如果 $f(x) = O(\sqrt{x})$，则

$$\int_0^{f(x)} e^{-u}\left(1 + \frac{u}{x}\right)^x du = \sqrt{2x}\int_0^{f(x)/\sqrt{2x}} e^{-t^2} dt + O(1)\ ]$$

▶ 9. [*HM36*] 对于很大的 x，$\gamma(x+1,px)/\Gamma(x+1)$的特性是什么？（这里 p 为实常数；且如果 $p<0$，我们假定 x 是一个整数使得对于负的 t，t^x 也是明确定义的。）在借助于 O 的项之前，试求得渐近展开的至少两项。

10. [*HM34*] 在上一题的假定下，且 $p\neq 1$，试求出对于固定的 y，$\gamma(x+1,px+py/(p-1))-\gamma(x+1,px)$的渐近展开，求到和上一题所得到的相同阶的项。

▶ 11. [*HM35*] 让我们通过引进一个参数 x 来推广函数 $Q(n)$和 $R(n)$：

$$Q_x(n)=1+\frac{n-1}{n}x+\frac{n-1}{n}\frac{n-2}{n}x^2+\cdots$$

$$R_x(n)=1+\frac{n}{n+1}x+\frac{n}{n+1}\frac{n}{n+2}x^2+\cdots$$

试剖析其中状况并求出当 $x\neq 1$ 时的渐近公式。

12. [*HM20*] 同正态分布（参见 1.2.10 小节）相关联而出现的函数 $\int_0^x e^{-t^2/2}\,dt$ 可以表达为不完全伽玛函数的一个特殊情况。试求 a，b 和 y 的值使得 $b\gamma(a,y)$ 等于 $\int_0^x e^{-t^2/2}dt$ 。

13. [*HM42*] (S. Ramanujan) 证明 $R(n)-Q(n)=\frac{2}{3}+8/(135(n+\theta(n)))$，其中 $\frac{2}{21}\leqslant\theta(n)\leqslant\frac{8}{45}$。（这隐含了更弱得多的结果 $R(n+1)-Q(n+1)<R(n)-Q(n)$。）

▶ 14. [*HM39*] (N. G. de Bruijn) 本题的目的是求对于固定的 α，当 $n\to\infty$ 时 $\sum_{k=0}^{n}k^{n+\alpha}e^{-k}$的渐近展开。

a) 以 $n-k$ 代替 k，证明给定的和等于

$$n^{n+\alpha}e^{-n}\sum_{k=0}^{n}e^{-k^2/2n}f(k,n)$$

其中

$$f(k,n)=(1-\frac{k}{n})^{\alpha}\exp(-\frac{k^3}{3n^2}-\frac{k^4}{4n^3}-\cdots)$$

b) 证明对于所有量 $m\geqslant 0$ 和 $\epsilon>0$，量 $f(k,n)$可以写成

$$\sum_{0\leqslant i\leqslant j\leqslant m}c_{ij}k^{2i+j}n^{-i-j}+O(n^{(m+1)(-1/2+3\epsilon)})\text{，若 }0\leqslant k\leqslant n^{1/2+\epsilon}$$

的形式。

c) 作为 b)的一个推论，证明对于所有 $\delta>0$，

$$\sum_{k=0}^{n}e^{-k^2/2n}f(k,n)=\sum_{0\leqslant i\leqslant j\leqslant m}c_{ij}n^{-i-j}\sum_{k\geqslant 0}k^{2i+j}e^{-k^2/2n}+O(n^{-m/2+\delta})$$

[提示：在 $n^{1/2+\epsilon}<k<\infty$ 变程上，对于所有的 r，此和为 $O(n^{-r})$。]

d) 证明对于固定的 $t\geqslant 0$，$\sum_{k\geqslant 0}k^t e^{-k^2/2n}$的渐近展开可由欧拉求和公式得到。

e) 因此最后

$$\sum_{k=0}^{n}k^{n+\alpha}e^{-k}=n^{n+\alpha}e^{-n}\left(\sqrt{\frac{\pi n}{2}}-\frac{1}{6}-\alpha+\left(\frac{1}{12}+\frac{1}{2}\alpha+\frac{1}{2}\alpha^2\right)\sqrt{\frac{\pi}{2n}}+O(n^{-1})\right)$$

对于任何希望的 r 值，这个计算原则上可推广到 $O(n^{-r})$。

15. [*HM20*] 证明下列积分与 $Q(n)$有关：

$$\int_0^{\infty}(1+\frac{z}{n})^n e^{-z}dz$$

16. [*M24*] 证明恒等式，当 $n>0$ 时，

$$\sum_k (-1)^k \binom{n}{k} k^{n-1} Q(k) = (-1)^n (n-1)!$$

17. [*HM29*] (K. W. Miller) 对称性要求我们也考虑第四个级数，如同 $R(n)$ 是相对于 $Q(n)$ 的那样，它是同 $P(n)$ 对称的：

$$S(n) = 1 + \frac{n}{n+1} + \frac{n}{n+2}\frac{n+1}{n+2} + \cdots = \sum_{k\geq 0} \frac{(n+k-1)!}{(n-1)!(n+k)^k}$$

这个函数的渐近特性如何？

18. [*M25*] 证明借助于 Q 函数可以非常简单地表达和式 $\sum\binom{n}{k}k^k(n-k)^{n-k}$ 和 $\sum\binom{n}{k}(k+1)^k(n-k)^{n-k}$。

19. [*HM30*] (Watson 引理) 证明对于所有很大的 n，如果 $C_n = \int_0^\infty e^{-nx} f(x)\,dx$ 存在，而且对于 $0 \leq x \leq r$，如果 $f(x) = O(x^\alpha)$，其中 $r>0$ 和 $\alpha > -1$，则 $C_n = O(n^{-1-\alpha})$。

▶ 20. [*HM30*] 假设和在(12)中一样，$u = w + \frac{1}{3}w^2 + \frac{1}{36}w^3 - \frac{1}{270}w^4 + \cdots = \sum_{k=1}^\infty c_k w^k$，是方程 $w = (u^2 - \frac{2}{3}u^3 + \frac{2}{4}u^4 - \frac{2}{5}u^5 + \cdots)^{1/2}$ 的幂级数解。证明对于所有 $m \geq 1$，

$$Q(n) + 1 = \sum_{k=1}^{m-1} k c_k \Gamma(k/2) \left(\frac{n}{2}\right)^{1-k/2} + O(n^{1-m/2})$$

[提示：应用 Watson 引理于习题 15 的恒等式。]

I feel as if I should succeed in doing something in mathematics,
although I cannot see why it is so very important.
尽管我不知道为什么它是那么重要，
但我觉得我应该在数学上有所作为。
—Helen Keller (1898)

1.3 MIX

本书自始至终有许多地方都需要引用一种计算机的内部机器语言。我们所用的计算机是称做“MIX”的一台虚构的计算机。MIX 除了也许更精巧外,同 20 世纪 60 年代和 70 年代的几乎每台计算机都非常相像。我们已把 MIX 的语言设计成足够地强有力,以便对于大多数算法都可以写出简短的程序,而且它又足够地简单,使得它的操作很容易学。

希望读者仔细地来学习这一小节。因为在本书的许多部分都有 MIX 语言出现。对于学习一种机器语言,应当是毫不犹豫的;其实,作者曾经发现,在一个星期之内,以六种不同的机器语言来写程序,并非不平常的事!凡对计算机不是仅仅一时感兴趣的每个人大概或早或晚都将需要掌握至少一种机器语言。MIX 已被专门地设计成保持历来的计算机的那些最简单方面,使得它的特征可以很容易地被理解。

然而,必须承认,MIX 现在已变得十分陈旧。因此,在本书的下一版本中,MIX 将被叫做 MMIX,即 2009 的新机器所代替。MMIX 将是一台所谓的精简指令系统的计算机(RISC)。它将对字长 64 位的字进行算术运算,它将比 MIX 更精巧,而且它将类似于在 20 世纪 90 年代占支配地位的那些计算机。

把本书中的内容都从 MIX 转换成 MMIX 将花很长时间。我们招聘志愿者来帮助进行转换的工作。同时,作者也希望人们愿意同老式的 MIX 结构一起多呆几年——MIX 仍然是值得了解的,因为它有助于弄清随后的发展的来龙去脉。

1.3.1 MIX 的描述

MIX 是世界上第一台多不饱和的(polyunsaturated)计算机。类似于大多数的机器,它有一个标识号码——1009。这个号码是通过取 16 种非常类似于 MIX 的(它们也可容易地模拟 MIX)真实的计算机,然后以相同的权对它们的代号作平均得来的。

$$\lfloor(360 + 650 + 709 + 7070 + U3 + SS80 + 1107 + 1604 + G20 + B220 + S2000 + 920 + 601 + H800 + PDP-4 + II)/16\rfloor = 1009 \tag{1}$$

通过采用罗马数字,亦可更简单地得到这个数——MIX*。

MIX 有一个独特的性质,即它同时既是二进制的,又是十进制的。*MIX 的程序员不必实际地知道他们是正在以二进制还是正在以十进制来对一台机器进行程序设计。*因此以 MIX 写成的算法只需稍作改动,就可以为每一种类型的机器所使用,而且也可以在每一种类型的机器上很容易地模拟 MIX。习惯于二进制计算机的程序员可以把 MIX 想像成是二进制的,而习惯于十进制的程序员可以把 MIX 想像成是十进制的。来自别

* 在罗马数字中,M 为 1000,IX 为 9。——译者注

的星球的程序员也可能选择把 MIX 想像为三进制的计算机。

字 信息的基本单位是字节。每个字节包含不确定数量的信息，但它必须能保存至少 64 个不同的值。即是说，我们知道，0 与 63 之间的任何数（包括 0 和 63 在内）都可包含在一个字节内。其次每个字节至多含有 100 个不同的值。在一台二进制的计算机上，一个字节因而必须由六个二进制位组成。在一台十进制的计算机上，每个字节我们有两位十进数字。*

以 MIX 语言表达的程序应该被写成，一个字节所具有的值不会超过 64 个。如果我们希望处理数 80，我们总应该保留两个相邻的字节来表达它，尽管对于一台十进制计算机来说一个字节也就足够了。*以 MIX 写成的算法应当正确地工作，而不论一个字节是多大。*尽管十分可能写依赖于字节大小的程序，但这样的行动同本书的精神是完全相违的；仅有的合法的程序是对于所有字节大小都将给出正确结果的那些程序。通常遵守这个根本规则并不难，我们将发现，对一台十进制计算机进行程序设计同对一台二进制计算机进行程序设计，并没有多大的差异。

两个相邻的字节可以表达数 0 到 4 095。

三个相邻的字节可以表达数 0 到 262 143。

四个相邻的字节可以表达数 0 到 16 777 215。

五个相邻的字节可以表达数 0 到 1 073 741 823。

*一个计算机字由五个字节和一个符号组成。*符号部分仅有两个可能的值，即 + 和 − 。

寄存器 MIX 中共有 9 个寄存器（参见图 13）：

A 寄存器（累加器）由五个字节和一个符号组成。

X 寄存器（扩充）类似地，也由五个字节和一个符号组成。

I 寄存器（变址寄存器）I1，I2，I3，I4，I5 和 I6 各拥有两个字节和一个符号。

J 寄存器（转移地址）拥有两个字节，它的符号总为 + 。

我们将在寄存器名字前边冠以“r”来标识一个 MIX 寄存器。因此“rA”指“寄存器A”。

A 寄存器有许多用途，特别是用于算术运算和对于数据的操作。X 寄存器是 rA“右边”的扩充，它同 rA 相关联，用于存放一个乘积或被除数的十个字节，或者用来存放由 A 向右移出的信息。变址寄存器 rI1，rI2，rI3，rI4，rI5 和 rI6 主要用于计数和访问可变的内存地址。J 寄存器总是保存最新的“转移”操作之后的指令的地址。因此它主要是同子程序相关联时使用。

除了它的寄存器外，MIX 包含

---

* 自 1975 年左右以来，“字节”一词已经用来表示恰有八个二进位的序列，它能够表示数 0 到 255。现实世界的字节因此要比假想的 MIX 机器的字节要大。其实 MIX 的老式的字节仅仅略大于现代的半字节。当我们在同 MIX 相关联而谈及字节时，我们将限定是在该词原先的含义下，折回到字节还未被标准化的年代。——原注

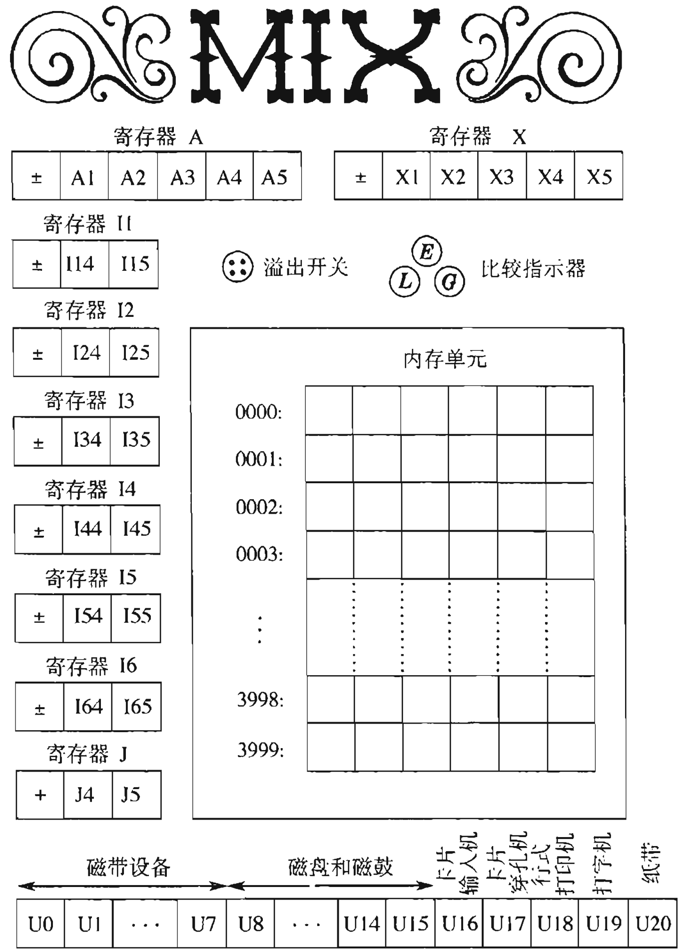


图 13　MIX 计算机

一个溢出开关(表示或“开”或“闭”的一位)；
一个比较指示器(有三个值：LESS(小于)，EQUAL(等于)或 GREATER(大于)；
内存(4000 字的存储单元，每个字有五个字节和一个符号)；
以及输入输出设备(卡片，磁带，磁盘等等)。

字的部分字段　一个计算机字的五个字节和符号编号如下：

0	1	2	3	4	5
±	字节	字节	字节	字节	字节

(2)

大多数指令允许程序员随意仅使用一个字的一部分。在这样的情况下可以给出一个标

准的"字段描述"。允许的字段是在一个计算机字中相邻的那些字段,它们用(L:R)来表示,其中L是字段左边的编号而R是字段右边的编号,字段描述的例子有:

(0:0),仅是符号位。

(0:2),符号位和头两个字节。

(0:5),整个字,这是最普通的字段说明。

(1:5),除了符号外的整个字。

(4:4),仅是第四个字节。

(4:5),两个最低有效字节。

字段描述的用法随指令的不同而略有不同,在用到它的每个指令时,都将予以详细说明。每个字段描述(L:R)在机器内部实际上由单个数8L+R来表示;注意这个数很容易放进一个字节中。

指令格式 用作指令的计算机字有下列形式:

0	1	2	3	4	5
±	A	A	I	F	C

(3)

最右边的字节C,是指出执行的是什么操作的操作码。例如,C=8指明操作LDA,即"装入A寄存器"。

F字节保存对于操作码的修改。它通常是一个字段说明(L:R)=8L+R;例如,如果C=8和F=11,则这个操作是"以(1:3)字段装入A寄存器"。有时F也用于其它目的;例如,对于输入输出指令,F是相关的输入或输出设备编号。

指令的左边部分±AA是地址。(注意符号是地址的一部分。)I字段紧接在地址之后,是变址说明,它可用来修改有效地址。如果I=0,地址±AA不作改动地被使用,否则I应包含1和6之间的一个数i,而且在实现指令的操作之前,变址寄存器Ii的内容被代数地加到±AA上;结果被用作地址。这个变址过程在每个指令上发生。我们将使用字母M来指示在特定的变址已经出现之后的地址。(如果变址寄存器和地址±AA相加产生出在两字节中放不下的一个结果,则M的值是无定义的。)

在大多数指令中,M将指一个内存单元。在本书中"内存单元"和"内存位置"几乎可交换地使用。我们假定,有4000个内存单元,其编号为从0到3999;因此每个内存单元都可用两个字节来编址。对于M指的是一个内存单元的每一个指令,我们必须有$0\leq M\leq 3999$,而且在此情况下,我们将写CONTENTS(M)来表示在内存位置M中存放的值。

在某些指令上,"地址"M有另外的意义,并且它可能是负的。因此一个指令把M加到一个变址寄存器上,这样一个操作要考虑M的符号。

记号 为了以易于阅读的方式来讨论指令,我们将使用记号

OP ADDRESS, I(F) (4)

来表示类似于(3)的一条指令。这里OP是对指令的操作码(C部分)所给出的一个符号名;ADDRESS是±AA部分;I和F分别表示I字段和F字段。

如果 I 为零,则“,I”被省略。如果 F 是对于这个特定的操作符的正常的 F 描述,则“(F)”也不必写。对于几乎所有操作符的正常的 F 描述是(0:5),表示整个字。如果一个不同的 F 是正常情况,则当讨论该具体的操作符时这一个点将被明确地提及。

例如装入一个数到累加器中的指令叫做 LDA,它的操作码是 8。我们有

约定的表示	实际的数值指令				
LDA 2000,2(0:3)	+	2000	2	3	8
LDA 2000,2(1:3)	+	2000	2	11	8
LDA 2000(1:3)	+	2000	0	11	8
LDA 2000	+	2000	0	5	8
LDA -2000,4	-	2000	4	5	8

(5)

指令“LDA 2000,2(0:3)”可以读作“以单元 2000 变址 2,将 0 至 3 字段的内容装入 A 中”。

为了表示一个 MIX 字的数值的内容,我们将总是使用像上边那样的方框记号,注意在字

+	2000	2	3	8

中,数 +2000 以填满两个相邻字节和符号示出;字节(1:1)的实际内容同字节(2:2)的实际内容将随着 MIX 计算机的不同而不同,因为字节大小是可变的。作为对于 MIX 字的这个记号的进一步的例子,框图

-	10000	3000

表示有两个字段的一个字,包含 10000 的 3 字节加符号的字段和包含 3000 的两字节字段。当把一个字分成一个以上字段时,就说它是被“组装”的。

每条指令的规则 上边在(3)之后的论述,已经对于用做指令的每个字的量 M,F 和 C 作了定义。现在我们将定义对应于每个指令的动作。

装入操作符

• LDA(装入 A),C = 8;F = 字段。

CONTENTS(M)的指定字段代替寄存器 A 以前的内容。

对于一个部分字段被用做一个输入的所有操作,如果符号被用做字段的一部分则使用之,否则把它理解为符号 +。当它被装入时,这个字段移到寄存器的右边部分。

例子:如果 F 是正常的字段说明(0:5),单元 M 中的每样东西都被复制到 rA 中。如果 F 是(1:5),则 CONTENTS(M)的绝对值连同正号被装入到 A。如果 M 包含一个指令字,且如果 F 是(0:2),则“± AA”字段被装入成

±	0	0	0	A	A

假设单元 2000 包含字

−	80	3	5	4

(6)

则从装入各种部分字段我们得到以下的结果：

指令	执行之后 rA 的内容					
LDA 2000	−	80		3	5	4
LDA 2000(1:5)	+	80		3	5	4
LDA 2000(3:5)	+	0	0	3	5	4
LDA 2000(0:3)	−	0	0	80		3
LDA 2000(4:4)	+	0	0	0	0	5
LDA 2000(0:0)	−	0	0	0	0	0
LDA 2000(1:1)	+	0	0	0	0	?

(最后一个例子有部分未知的效果，因为字节大小是可变的。)

• LDX(装入 X)。C = 15；F = 字段。

除了被装入的是 rX 而不是 rA 之外，这个指令和 LDA 相同。

• LDi(装入 i)。C = 8 + i，F = 字段。

除了被装入的是 rIi 而不是 rA 之外，这个指令和 LDA 相同。一个变址寄存器仅含两个字节(而不是五个)和一个符号；总是假定字节 1,2,3 为 0。如果它的结果是置字节 1,2,3 为 0 之外的任何值，则 LDi 指令无定义。

在所有指令的描述中，“i”代表一个整数，$1 \leqslant i \leqslant 6$，因此，LD$i$ 代表六个不同的指令：LD1，LD2，…，LD6。

• LDAN(把负的装入 A)。C = 16；F = 字段。

• LDXN(把负的装入 X)。C = 23；F = 字段。

• LDiN(把负的装入 i)。C = 16 + i；F = 字段。

除相反的符号被装入之外，这八个指令分别和 LDA，LDX，LDi 相同。

存储操作符

• STA(存 A)。C = 24；F = 字段。

rA 内容的一部分代替由 F 所确定的 CONTENTS(M)的某个字段。CONTENTS(M)的其余部分不变。

对于一个*存储*操作，字段 F 有和*装入*操作相反的意义：字段中的字节数是从寄存器右边部分取的，而且如有必要就左移以插入到 CONTENTS(M)的适当字段中。符号不改变，除非它是字段的一部分。寄存器的内容不受影响。

例子:假设单元 2000 含

-	1	2	3	4	5

而寄存器 A 包含

+	6	7	8	9	0

则:

指令	执行之后单元 2000 的内容					
STA　2000	+	6	7	8	9	0
STA　2000(1:5)	-	6	7	8	9	0
STA　2000(5:5)	-	1	2	3	4	0
STA　2000(2:2)	-	1	0	3	4	5
STA　2000(2:3)	-	1	9	0	4	5
STA　2000(0:1)	+	0	2	3	4	5

• STX(存 X)。C = 31;F = 字段。

除所存的是 rX 而不是 rA 之外,和 STA 一样。

• STi(存 i)。C = 24 + i;F = 字段。

除所存储的是 rIi 而不是 rA 之外,和 STA 一样。一个变址寄存器的字节 1,2,3 为 0;因此如果 rI1 含

±	m	n

则它就像是

±	0	0	0	m	n

那样。

• STJ(存 J)。C = 32;F = 字段。

除所存的是 rJ 而且其符号总为 + 外,和 STi 相同。

对于 STJ,F 的正常字段描述是(0:2),*而不是*(0:5)。这是自然的,因为 STJ 几乎总是对一条指令的地址字段来操作的。

• STZ(存零)。C = 33;F = 字段。

除所存的是正零之外,和 STA 一样。换言之,CONTENTS(M)的指定字段被清零。

算术运算符　对加法,减法,乘法,除法运算,都允许有字段描述。"(0:6)"的字段描述可用于表示一个"浮点"运算(参见 4.2 节),但我们对于 MIX 将要写出的程序很少使用这一特征,因为我们主要将涉及关于整数的算法。

和通常一样,标准的字段描述是(0:5)。其余的字段描述如同在 LDA 中那样处理。我们将使用字母 V 表示 CONTENTS(M)特定的字段。因此,如果操作码是 LDA,则 V 是

装到寄存器 A 的内容。

• ADD(加法)。C = 1;F = 字段。

V 被加到 rA 上。如果对于寄存器 A 来说结果的数值太大,则置溢出开关,在 rA 中出现的是加法的余数,而想像一个"1"已进位到寄存器 A 左边的另一个寄存器。(否则溢出开关的设定不变。)如果结果为零,则 rA 的符号不变。

例子:以下的指令序列计算寄存器 A 的五个字节之和。

```
STA 2000
LDA 2000(5:5)
ADD 2000(4:4)
ADD 2000(3:3)
ADD 2000(2:2)
ADD 2000(1:1)
```

这有时叫做"横加"。

在某些 MIX 计算机上将出现溢出而在其它的 MIX 计算机上却并不出现,这是由于字节大小可变的定义所致。我们并未说过如果值大于 1073741823 溢出肯定会出现。依赖于字节的大小,当结果的数值超过五个字节的内容时才发生溢出。不管字节大小如何,人们仍能写出正确地工作并给出同样的最终答案的程序来。

• SUB(减法)。C = 2;F = 字段。

从 rA 减去 V 的值。(等价于 ADD 但以 - V 替代 V。)

• MUL(乘法)。C = 3;F = 字段。

10 个字节的乘积,V 乘上 rA,替代寄存器 A 和 X 原来的值。rA 和 rX 的符号都被置成乘积的代数符号(即,如果 V 和 rA 的符号相同,则为 + ,如果它们不同则为 -)。

• DIV(除法)。C = 4;F = 字段。

把 rA 和 rX 的值作为以 rA 的符号为符号的 10 个字节的数 rAX,除之以 V 的值。如果 V = 0 或如果商的数值多于五个字节(这等于条件 $|rA| \geq |V|$),就以不确定的信息填满寄存器 A 和 X 并置溢出开关。否则把商 $\pm \lfloor |rAX/V| \rfloor$ 放置于 rA 中而把余数 $\pm(|rAX| \bmod |V|)$ 放置于 rX 中。之后 rA 的符号是商的代数符号(即是,如果 V 和 rA 的符号相同则为 + ,如果它们不同则为 -)。之后 rX 的符号是原来的 rA 的符号。

算术指令的例子:在大多数情况下,算术运算仅对单个五字节数的 MIX 字进行。这些数不被组装为若干个字段。然而,如果谨慎从事,仍有可能对组装了的 MIX 字进行算术操作。下列例子应予仔细研究。(和前边一样,? 表示一个未知的值。)

	+	1234	1	150	执行前的 rA
	+	100	5	50	单元 1000
ADD 1000	+	1334	6	200	执行后的 rA

SUB 1000

−	1234		0	0	9	执行前的 rA
−	2000		150		0	单元 1000
+	766		149		?	执行后的 rA

MUL 1000

+	1	1	1	1	1	执行前的 rA
+	1	1	1	1	1	单元 1000
+	0	1	2	3	4	执行后的 rA
+	5	4	3	2	1	执行后的 rX

MUL 1000(1:1)

−					112	执行前的 rA
?	2	?	?	?	?	单元 1000
−					0	执行后的 rA
−					224	执行后的 rX

MUL 1000

−	50	0	112		4	执行前的 rA
−	2	0	0	0	0	单元 1000
+	100		0	224		执行后的 rA
+	8	0	0	0	0	执行后的 rX

DIV 1000

+					0	执行前的 rA
?					17	执行前的 rX
+					3	单元 1000
+					5	执行后的 rA
+					2	执行后的 rX

DIV 1000

−					0	执行前的 rA
+	1235		0	3	1	执行前的 rX
−	0	0	0	2	0	单元 1000
+	0	617		?	?	执行后的 rA
−	0	0	0	?	1	执行后的 rX

(这些例子是以这样一个思路准备的，即给出完备的令人困惑的描述比不完备的直接了当的描述要更好些。)

地址传送操作符 在下列操作中，(可能带变址的)“地址”M被用作一个带符号的数，而不作为内存中一个单元的地址。

• ENTA (进入 A)。C = 48；F = 2。

把量M装入rA中。这个动作等价于从包含M的带符号的值的一个内存字执行“LDA”。如果M = 0，则装入指令的符号。

例子：“ENTA 0”把rA置成0，并有+的符号。“ENTA 0,1”把rA置成变址寄存器1当前的内容，但-0被变成+0。“ENTA -0,1”和这个类似，但+0被变成-0。

• ENTX (进入 X)。C = 55；F = 2。

• ENT*i* (进入 *i*)。C = 48 + *i*；F = 2。

和ENTA类似，但装入相应的寄存器。

• ENNA (进入负的 A)。C = 48；F = 3。

• ENNX (进入负的 X)。C = 55；F = 3。

• ENN*i* (进入负的 *i*)。C = 48 + *i*；F = 3。

除了装入相反的符号外，和ENTA，ENTX及ENT*i*相同。

例子：“ENN3 0,3”以rI3负的值代替rI3，只是-0仍保留为-0。

• INCA (A 增值)。C = 48；F = 0。

把量M加到rA中；这个动作等价于从包含M的值的一个内存字执行“ADD”。可能出现溢出，其处理就如在ADD中一样。

例子：“INCA 1”使rA的值加1。

• INCX (X 增值)。C = 55；F = 0。

把量M加到rX中。如果出现溢出，则这个动作等价于ADD，只是使用rX来代替rA。这一指令对于寄存器A绝无影响。

• INC*i* (*i* 增值)。C = 48 + *i*；F = 0。

把M加到rI*i*中。一定不能出现溢出；如果M + rI*i*不能装入两个字节中，则本指令的结果无定义。

• DECA (A 减值)。C = 48；F = 1。

• DECX (X 减值)。C = 55；F = 1。

• DEC*i* (*i* 减值)。C = 48 + *i*；F = 1。

这八条指令分别和INCA，INCX以及INC*i*相同，只是从寄存器减去M而不是加上M。

注意操作码C对ENTA，ENNA，INCA以及DECA是一样的；使用F字段来区分各种不同的操作。

比较操作符 MIX的比较操作符总是比较一个寄存器中的值和内存中的一个值。按照寄存器的值是否小于，等于或大于内存单元的值，来把比较指示符置成LESS，EQUAL或GREATER。负的零等于正的零。

• CMPA（比较 A）。C = 56；F = 字段。

rA 的特定字段同 CONTENTS(M)相同字段的值进行比较。如果 F 字段不包括符号位，则两个字段都被认为是非负的；否则在比较中要把符号考虑在内。（当 F 是(0:0)时，将总是出现相等，因为负零等于正零。）

• CMPX（比较 X）。C = 63；F = 字段。

类似于 CMPA。

• CMPi（比较 i）。C = 56 + i；F = 字段。

类似于 CMPA。在比较中，变址寄存器的字节 1,2 和 3 被当做零。（因此，如果 F = (1:2)，结果不可能是 GREATER。）

转移操作符　通常指令都是顺序执行的；换言之，在单元 P 中的指令被执行过后接着执行的指令通常是在单元 P + 1 中找到的指令。但是有若干个“转移”指令允许这一顺序被中断。当出现一个典型的转移时，把 J 寄存器置为下一指令的地址（即，如果我们未转移，下一个要被执行的指令的地址）。然后，如果需要，程序员可以使用“存 J”指令来设置将被使用的另一个指令的地址字段以便返回程序原来的位置。每当一个转移真正地出现于一个程序中时，J 寄存器就被改变，除非转移操作符是 JSJ，而且 J 寄存器绝不会为无转移所改变。

• JMP（转移）。C = 39；F = 0。

无条件转移，下一条指令取自于单元 M。

• JSJ（转移，保持 J 不变）。C = 39；F = 1。

除了 rJ 的内容不变之外，和 JMP 相同。

• JOV（溢出时转移）。C = 39；F = 2。

如果溢出开关置位，则把它复位并出现一个转移(JMP)，否则什么事情也不发生。

• JNOV（无溢出时转移）。C = 39；F = 3。

如果溢出开关复位，则出现一个转移，否则将它复位。

• JL，JE，JG，JGE，JNE，JLE（小于转移，相等时转移，大于转移，大于等于转移，不相等时转移，小于等于转移）。C = 39；分别地 F = 4,5,6,7,8,9。

如果比较指示器被设置成所指出的条件，则转移。例如，如果比较指示器是 LESS 或 GREATER 则 JNE 将转移。这些指令不改变比较指示器的设定。

• JAN，JAZ，JAP，JANN，JANZ，JANP（A 为负时转移，A 为零时转移，A 为正时转移，A 非负时转移，A 非零时转移，A 非正时转移）。C = 40；分别地 F = 0,1,2,3,4,5。

如果 rA 的内容满足所述条件，则出现一个转移，否则什么事情也不发生。“正”意味着大于零（非零），“非正”则相反，即指零或负。

• JXN，JXZ，JXP，JXNN，JXNZ，JXNP（X 为负时转移，X 为零时转移，X 为正时转移，X 为非负时转移，X 为非零时转移，X 为非正时转移）。C = 47；分别地 F = 0,1,2,3,4,5。

• JiN，JiZ，JiP，JiNN，JiNZ，JiNP（i 为负时转移，i 为零时转移，i 为正时转移，i 为非负时转移，i 为非零时转移，i 为非正时转移）。C = 40 + i；分别地 F = 0,1,2,3,4,5。这些指令类似于对于 rA 相应的操作。

其它操作符

• SLA, SRA, SLAX, SRAX, SLC, SRC (A 左移, A 右移, AX 左移, AX 右移, AX 循环左移, AX 循环右移)。C = 6; 分别地 F = 0,1,2,3,4,5。

这些是"移位"指令,其中 M 指定向左移或向右移的 MIX 的字节个数; M 必须是非负的。SLA 和 SRA 不影响 rX。其它的移位影响寄存器 A 也影响寄存器 X,似乎 A 和 X 是单个 10 字节的寄存器。对于 SLA, SRA, SLAX 和 SRAX,在一边把 0 移进寄存器中。而在另一边移出的字节消失。指令 SLC 和 SRC 要求循环移位,其中在一端移出的字节进入到另一端。rA 和 rX 两个寄存器都参与到循环移位当中,任何移位指令都不影响寄存器 A 和 X 的符号。

例子:

	寄存器 A						寄存器 X					
初始内容	+	1	2	3	4	5	−	6	7	8	9	10
SRAX 1	+	0	1	2	3	4	−	5	6	7	8	9
SLA 2	+	2	3	4	0	0	−	5	6	7	8	9
SRC 4	+	6	7	8	9	2	−	3	4	0	0	5
SRA 2	+	0	0	6	7	8	−	3	4	0	0	5
SLC 501	+	0	6	7	8	3	−	4	0	5	0	0

• MOVE(移动)。C = 7; F = 数。

由 F 所指定的字数被移动,由单元 M 开始移动到由变址寄存器 1 的内容所指定的单元。一次传送一个字,在本操作结束时 rI1 的值加上了 F 的值。但如果 F = 0,则什么事情也不做。标准的 F 为 1。

当涉及的单元组相重叠时应小心;例如,假设 F = 3 而 M = 1000。那么,若 rI1 = 999,我们把 CONTETS(1000)传送到 CONTENTS(999),把 CONTENTS(1001)传送到 CONTENTS(1000),以及 CONTENTS(1002)传送到 CONTENTS(1001);这里不出现什么异常情况。但若 rI1 是 1001,那我们将把 CONTENTS(1000)传送到 CONTENTS(1001),然后 CONTENTS(1001)传送到 CONTENTS(1002),然后 CONTENTS(1002)传送到 CONTENTS(1003),所以我们已经把同一个内容 CONTENTS(1000)传送到三个地方。

• NOP (空操作)。C = 0。

无操作出现,因此绕过本指令,忽略 F 和 M 的值。

• HLT (停机)。C = 5; F = 2。

机器停机。当计算机操作员重新启动它时,其实际效果等价于 NOP。

输入输出操作符 MIX 有相当数量的输入输出设备(每一种设备都是以额外的费用任选的)。对每一个设备赋予一个号码如下:

设备号	外围设备	块大小
t	磁带设备号 t $(0\leq t\leq 7)$	100 字
d	磁盘或磁鼓设备号 d $(8\leq d\leq 15)$	100 字
16	卡片输入机	16 字
17	卡片穿孔机	16 字
18	行式打印机	24 字
19	打字机终端	14 字
20	纸带	14 字

并非每个 MIX 装置都有所有这些设备可用;我们将偶尔地适当假设某些设备的存在。某些设备不可以既用作输入又用作输出。上面列出的字数是同每一个设备相关联的固定块大小。

使用磁带、磁盘或磁鼓设备进行的输入或输出读或写完整的字(五个字节加上一个符号)。然而,使用号码为 16 ~ 20 的设备进行输入或输出总是以字符代码进行的,其中每个字节表示一个字符。因此,每个 MIX 字要传送五个字符。表 1 的顶部给出了字符的代码(表 1 出现于本节的末尾和本书的最后几页处)。代码 00 对应于"␣",它表示一个空格。代码 01 ~ 29 代表字母 A 到 Z 连同插入的几个希腊字母。代码 30 ~ 39 表示数字0,1,…,9;而再往后的代码 40,41,…表示标点符号以及其它的特殊符号。(MIX 的字符集回溯到计算机能够处理小写字母以前的岁月。)我们不能使用字符代码来读入或写出一个字节可能有的所有值,因为某些组合是无定义的。而且,某些输入输出设备可能不能处理字符集中的所有符号;例如,出现在字母中间的Σ和Π符号也许对于卡片输入机就是不可接受的。当进行字符代码输入时,所有字的符号都被设置为 + ;在输出时,符号被忽略。如果一个打字机被用作输入,则在每行末尾敲入的回车使得该行的剩余位置都被填上空格。

磁盘和磁鼓设备是外部存储设备,它们各含 100 个字的块。在以下所定义的每个 IN,OUT 或 IOC 指令中,由该指令访问的具体的 100 字块由 rX 的当前内容所确定,它不应超出所涉及的磁盘或磁鼓的容量。

• IN (输入)。C = 36;F = 设备。

这一指令启动由特定的输入设备到以 M 开始的连续的单元的信息传送,所传送的单元数是这个设备的块大小(参见上述的表)。如果同一个设备的前边一个操作还未完成,机器就将在这点上等候。由这一指令开始的信息传送,经过一段未知的时间后才能完成。这个时间依赖于该输入设备的速度。所以一个程序在此期间不得访问内存中的该信息。试图从磁带读取刚刚写到该带上最新块之后的任何块,也是不妥当的。

• OUT (输出)。C = 37;F = 设备。

这个指令启动从以 M 开始的内存单元到所指定的输出设备的信息传送。如果该输出设备还未准备好,则机器等候到它准备好为止。这个传送在某个未知的将来的时刻前将一直不能完成,这个时间依赖于该输出设备的速度。所以一个程序在此期间不得修改内存中的这个信息。

• IOC（输入输出控制）。C = 35；F = 设备。

如果必要，机器等候，直到所确定的设备不再忙碌为止。然后依赖于所使用的具体设备，实施一个控制操作。下列例子被应用于本书的各个部分。

磁带：如果 M = 0，磁带被反绕。如果 M < 0，磁带向后跳过 M 个块，或跳到磁带的开始处，看哪一种情况先出现。如果 M > 0，磁带就向前跳；跳过最后所写的块之后的任何块上，都是不妥当的。

例如，序列“OUT 1000 (3)；IOC -1(3)；IN 2000 (3)”写出一百个字到带 3 上，然后再把它读回来。除非磁带的可靠性有问题，上边这个序列的后两条指令是把字 1000 ~ 1099 传送到单元 2000 ~ 2099 的很慢的方法。指令序列“OUT 1000 (3)；IOC +1 (3)”是不适当的。

磁盘或磁鼓：M 应为零。其效果是按照 rX 来对设备定位，使得在这个设备上的下一个 IN 或 OUT 操作如果仍使用相同的 rX 的设定，可以少花些时间。

行式打印机：M 应为零。“IOC 0 (18)”使打印机跳到下一页纸的顶部。

纸带输入机：M 应为零。“IOC 0 (20)”反绕纸带。

• JRED（就绪转移）。C = 38；F = 设备。

如果所指定的设备已准备就绪，则出现转移，即，完成由 IN，OUT 或 IOC 启动的上一个操作。

• JBUS（忙碌转移）。C = 34；F = 设备。

类似于 JRED，但当所指定的设备未准备就绪时出现转移。

例子：在单元 1000 中，指令“JBUS 1000 (16)”将反复地被执行直到设备 16 就绪为止。

以上的简单操作就是 MIX 的全部输入输出指令。这里没有“带校验”指示器等，来报告在外部设备上的异常情况。任何这样的情况（例如纸塞住、设备断电、没有磁带等等）会引起设备保持忙碌，响铃，而熟练的计算机操作员使用通常的维护步骤人工地进行修理。某些更为复杂的外围设备，它们比起这里所描述的固定块大小的磁带、磁鼓和磁盘来更为昂贵，更具有作为现代设备的代表性，将在 5.4.6 和 5.4.9 节讨论。

转换操作符

• NUM（转换成数值）。C = 5；F = 0。

这个操作用来把字符代码变为数值代码。M 被忽略。假定寄存器 A 和 X 包含的是字符代码之下的一个 10 字节的数；NUM 指令就把 rA 的数值置成等于这个数的数值（视为十进数）。rX 的值和 rA 的符号不变。字节 00，10，20，30，40 转换成数字零；字节 01，11，21，…转换成数字 1 等等；有可能溢出。而在此情况下，保留模 b^5 的余数，其中 b 是字节大小。

• CHAR（转换成字符）。C = 5；F = 1。

这个操作用来把数值代码变成适合于对穿孔卡片机或磁带或行式打印机输出的字符代码。rA 中的值被转换成一个 10 字节十进制数，并以字符代码的形式放入寄存器 A

和 X 中。rA 和 rX 的符号不变。M 被忽略。

例子：

	寄存器 A						寄存器 X					
初始内容	-	00	00	31	32	39	+	37	57	47	30	30
NUM 0	-	12977700					+	37	57	47	30	30
INCA 1	-	12977699					+	37	57	47	30	30
CHAR 0	-	30	30	31	32	39	+	37	37	36	39	39

计时　为了给出 MIX 程序的效率的定量信息，对于每一 MIX 操作都指定了对于 1970 年制造的计算机来说典型的执行时间。

ADD，SUB，所有 LOAD 操作，所有 STORE 操作（包括 STZ 在内），所有移位指令，所有比较操作都花费 2 个单元的时间。MOVE 要求 1 个单元时间加上每移动一个字再花费 2 个单元时间。MUL，NUM，CHAR，各要求 10 个单元时间而 DIV 要求 12 个单元时间。浮点操作的执行时间在 4.2.1 小节中说明。所有剩下的其它操作花费 1 个单元时间，再加上在执行 IN，OUT，IOC 或 HLT 指令时计算机可能空闲的时间。

特别还要注意，ENTA 花费 1 个单元时间，而 LDA 花费 2 个单元时间。这些计时规则很容易记，因为除了移位、转换、MUL 和 DIV 之外，单元时间数等于访问内存的次数（包括访问指令本身在内）。

MIX 的基本时间单位是一个相对的测量单位，我们将简单地以 u 来表示它。它可以被认为是，比如说 10 毫秒（对于一台相对便宜的计算机来说），或 10 纳秒（对于一台相对高价的机器来说）。

例子：序列 LDA 1000；INCA 1；STA 1000 恰好花 $5u$ 的时间。

And now I see with eye serene
The very pulse of the machine.
因此现在我以平静的眼神
看到了机器的脉动。
—WILLIAM WORDSWORTH,
She Was a Phantom of Delight (1804)

小结　除了“GO 按钮”之外，我们现在已经讨论了 MIX 的所有特性。“GO 按钮”在习题 26 中讨论。尽管 MIX 有近乎 150 种不同的操作，但由于它们已分成了少数几种简单的类型，因此很容易记。表 1 综述了对于每个 C 的设定对应的操作。每个操作符的名称后边是用圆括弧括起来的默认的 F 字段。

下列习题提供对本小节内容的快速复习。它们大部分都十分简单，因此读者应当尝试把几乎所有题目都做一做。

习 题

1.[*00*] 如果MIX是一台三进制(基数为3)的计算机,则每个字节需要有多少“三进制”位?

2.[*02*] 如果在 MIX 中可表示的值可以大到 99999999,则为包含这一数量应当使用多少相邻的字节?

3.[*02*] 对于(a) 地址字段,(b) 变址字段,(c)字段的字段,以及(d) 一条 MIX 指令的操作码字段,试给出部分字段描述(L:R)。

4.[*00*] 在(5)中最后的例子是“LDA - 2000,4”。就内存地址不应是负的事实来看,这怎样才能合法?

5.[*10*] 如果把(6)当做一条 MIX 指令,则类似于(4),什么符号记号对应于(6)?

▶ 6.[*10*] 假定单元 3000 包含

+	5	1	200	15

下列指令的结果是什么?(指出这些指令当中有没有无定义的或仅仅部分地定义的。)

(a) LDAN 3000; (b) LD2N 3000(3:4); (c) LDX 3000(1:3);

(d) LD6 3000; (e) LDXN 3000(0:0)。

7.[*M15*] 使用代数操作 $X \bmod Y$ 和$\lfloor X/Y \rfloor$,试给出对于所有不出现溢出情况的 DIV 指令的结果的精确定义。

8.[*15*] 在 128 页上出现的 DIV 指令的最后一个例子有“执行前的 rX”等于 | + | 1235 | 0 | 3 | 1 | 。如果代之以 | − | 1234 | 0 | 3 | 1 | ,而例子的其它部分不变,试问在 DIV 指令之后寄存器 A 和 X 将包含什么?

▶ 9.[*15*] 列出可能会影响溢出开关的设定的所有 MIX 操作符。(不要包括浮点操作符。)

10.[*15*] 列出可能会影响比较指示器的设定的所有 MIX 操作符。

▶ 11.[*15*] 列出可能会影响 rI1 的设定的所有 MIX 操作符。

12.[*10*] 试找出一个指令,它相当于以 2 来乘 rI3 中的内容和在 rI3 中保留结果。

▶ 13.[*10*] 假设单元 1000 包含指令“JOV 1001”。如果溢出开关置位,则这个指令把它复位(而且因此,在任何情况下,下一条欲执行的指令将在单元 1001 中)。如果把这一指令改变成“JNOV 1001”,则将有什么差别?如果把它改成“JOV 1000”或“JNOV 1000”,那又会怎样?

14.[*20*] 对于每个 MIX 操作,考虑是否有一种方法来设置 ± AA,I 和 F 部分使得指令的结果恰好等于 NOP(除了执行时间可能更长些之外)。假定对于任何寄存器的内容以及任何内存单元都毫无所知。只要有可能产生一个 NOP 即可,指出它是如何做成的。例子:如果地址和变址部分为零,INCA 是空操作。JMP 绝不是一个空操作,因为它影响 rJ。

15.[*10*] 一个打字机或纸带输入机的块有多少字母数字字符?一个卡片阅读机或卡片穿孔机块呢?一个行式打印机块中呢?

16.[*20*] 编写一个程序,它把内存单元 0000 ~ 0099 全置为零,而且(a)这个程序应尽可能地短;(b)这个程序的运行应尽可能地快。(提示:试考虑使用 MOVE 指令。)

表 1

字符	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
代码	⊔	A	B	C	D	E	F	G	H	I	Δ	J	K	L	M	N	O	P	Q	R	Σ	Π	S	T	U

00	*1*	**01**	*2*	**02**	*2*	**03**	*10*
无操作 NOP(0)		rA←rA + V ADD(0:5) FADD(6)		rA←rA − V SUB(0:5) FSUB(6)		rAX← rA × V MUL(0:5) FMUL(6)	
08	*2*	**09**	*2*	**10**	*2*	**11**	*2*
rA←V LDA(0:5)		rI1←V LD1(0:5)		rI2←V LD2(0:5)		rI3←V LD3(0:5)	
16	*2*	**17**	*2*	**18**	*2*	**19**	*2*
rA← − V LDAN(0:5)		rI1← − V LD1N(0:5)		rI2← − V LD2N(0:5)		rI3← − V LD3N(0:5)	
24	*2*	**25**	*2*	**26**	*2*	**27**	*2*
M(F)←rA STA(0:5)		M(F)←rI1 ST1(0:5)		M(F)←rI2 ST2(0:5)		M(F)←rI3 ST3(0:5)	
32	*2*	**33**	*2*	**34**	*1*	**35**	*1* + T
M(F)←rJ STJ(0:2)		M(F)←0 STZ(0:5)		设备 F 忙碌吗? JBUS(0)		控制,设备 F IOC(0)	
40	*1*	**41**	*1*	**42**	*1*	**43**	*1*
rA:0,转移 JA[+]		rI1:0,转移 J1[+]		rI2:0,转移 J2[+]		rI3:0,转移 J3[+]	
48	*1*	**49**	*1*	**50**	*1*	**51**	*1*
rA←[rA]? ± M INCA(0) DECA(1) ENTA(2) ENNA(3)		rI1←[rI1]? ± M INC1(0) DEC1(1) ENT1(2) ENN1(3)		rI2←[rI2]? ± M INC2(0) DEC2(1) ENT2(2) ENN2(3)		rI3←[rI3]? ± M INC3(0) DEC3(1) ENT3(2) ENN3(3)	
56	*2*	**57**	*2*	**58**	*2*	**59**	*2*
CI←rA(F):V CMPA(0:5) FCMP(6)		CI←rI1(F):V CMP1(0:5)		CI←rI2(F):V CMP2(0:5)		CI←rI3(F):V CMP3(0:5)	

一般形式

C	*t*
描述 OP(F)	

C = 操作码,指令的(5:5)字段

F = 操作码的变形,指令的(4:4)字段

M = 变址后的指令地址

V = M(F) = 单元 M 的 F 字段的内容

OP = 操作的符号名

(F) = 标准的 F 设定

t = 执行时间　　*T* = 互锁时间

25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
V	W	X	Y	Z	0	1	2	3	4	5	6	7	8	9	.	,	(	)	+	-	*	/	=	$	<	>	@	;	:	'

04	*12*	05	*10*	06	*2*	07	*1 + 2F*
rAX←rAX/V rX←余数 DIV(0:5) FDIV(6)		特殊的 NUM(0) CHAR(1) HLT(2)		移位 M 个字节 SLA(0) SRA(1) SLAX(2) SRAX(3) SLC(4) SRC(5)		从 M 移动 F 字 到 rI1 MOVE(1)	
12	*2*	13	*2*	14	*2*	15	*2*
rI4←V LD4(0:5)		rI5←V LD5(0:5)		rI6←V LD6(0:5)		rX←V LDX(0:5)	
20	*2*	21	*2*	22	*2*	23	*2*
rI4←－V LD4N(0:5)		rI5←－V LD5N(0:5)		rI6←－V LD6N(0:5)		rX←－V LDXN(0:5)	
28	*2*	29	*2*	30	*2*	31	*2*
M(F)←rI4 ST4(0:5)		M(F)←rI5 ST5(0:5)		M(F)←rI6 ST6(0:5)		M(F)←rX STX(0:5)	
36	*1 + T*	37	*1 + T*	38	*1*	39	*1*
输入,设备 F IN(0)		输出,设备 F OUT(0)		设备 F 就绪? JRED(0)		转移 JMP(0) JSJ(1) JOV(2) JNOV(3) 还有下边的[*]	
44	*1*	45	*1*	46	*1*	47	*1*
rI4:0,转移 J4[+]		rI5:0,转移 J5[+]		rI6:0,转移 J6[+]		rX:0,转移 JX[+]	
52	*1*	53	*1*	54	*1*	55	*1*
rI4←[rI4]? ± M INC4(0) DEC4(1) ENT4(2) ENN4(3)		rI5←[rI5]? ± M INC5(0) DEC5(1) ENT5(2) ENN5(3)		rI6←[rI6]? ± M INC6(0) DEC6(1) ENT6(2) ENN6(3)		rX←[rX]? ± M INCX(0) DECX(1) ENTX(2) ENNX(3)	
60	*2*	61	*2*	62	*2*	63	*2*
CI←rI4(F):V CMP4(0:5)		CI←rI5(F):V CMP5(0:5)		CI←rI6(F):V CMP6(0:5)		CI←rX(F):V CMPX(0:5)	

rA = 寄存器 A
rX = 寄存器 X
rAX = 寄存器 A 和 X 当做一个
r*I*i = 变址寄存器 *i*,1≤*i*≤6
rJ = 寄存器 J
CI = 比较指示器

[*]　　[+]:
JL(4) < N(0)
JE(5) = Z(1)
JG(6) > P(2)
JGE(7) ≥ NN(3)
JNE(8) ≠ NZ(4)
JLE(9) ≤ NP(5)

17. [*26*] 除了单元是 0000 到 N(包括 0000 和 N 在内),本题和上一题一样,是把它们全置为零,其中 N 是 rI2 当前的内容。你的程序(a)和(b)应当对于 $0 \leqslant N \leqslant 2999$ 的任何值都有效,它们应在单元 3000 处开始。

▶ 18. [*22*] 在以下的"第 1 号"程序执行过后,寄存器、开关和内存发生什么变化?(例如,rI1 最后的设定是什么? rX 呢? 溢出开关和比较指示器呢?)

```
STZ 1
ENNX 1
STX 1(0:1)
SLAX 1
ENNA 1
INCX 1
ENT1 1
SRC 1
ADD 1
DEC1 - 1
STZ 1
CMPA 1
MOVE - 1,1(1)
NUM 1
CHAR 1
HLT 1
```
▌

▶ 19. [*14*] 不计 HLT 指令在内,上题程序的执行时间是多少?

20. [*20*] 试编写一个程序,它把所有 4000 个内存单元都置成"HLT"指令,而后停止。

▶ 21. [*24*] (a) J 寄存器能为零吗? (b) 在 rI4 中给定一个数 N,并假 $0 < N \leqslant 3000$,试编写一个程序,它把寄存器 J 置为 N。你的程序应在单元 3000 处开始。当你的程序完成它的执行时,所有内存单元的内容必须不改变。

▶ 22. [*28*] 单元 2000 包含一个整数 X。试写出两个程序,它们计算 X^{13},停机时结果在寄存器 A 中。一个程序应该使用最少个数的 MIX 的内存单元,另一个程序应该使用最少的执行时间。假定 X^{13}可装入一个字中。

23. [*27*] 单元 0200 包含一个字

+	*a*	*b*	*c*	*d*	*e*

试写出两个程序,它们计算"自反的"字

+	*e*	*d*	*c*	*b*	*a*

停机时结果在寄存器 A 中。其中一个程序应当不使用 MIX 的装入和存储字的部分字段的能力来做这件事。在所指出的条件下,两个程序应当使用最少个数的内存单元(包括程序使用的所有单元以及中间结果所用的临时存储单元在内)。

24. [*21*] 假设寄存器 A 和 X 分别含

+	0	*a*	*b*	*c*	*d*

和

+	*e*	*f*	*g*	*h*	*i*

试写出两个程序,它们分别地使用(a) 最少的内存空间,(b) 最少的执行时间,来改变这两个寄存器的内容使成为

+	a	b	c	d	e

和

+	0	f	g	h	i

▶ 25. [*30*] 假设 MIX 的厂家希望产生出一台更强有力的计算机("Mixmaster"?),而且他要说服尽可能多的现在拥有 MIX 计算机者投资于这个更昂贵的机器上。他想把这个新机器的硬件设计成 MIX 的扩充,即对 MIX 正确地写成的所有程序无须任何改动就能在新的机器上工作。请给出关于可加入到这个扩充中合适的东西的建议。(例如:你能否更好地使用一条指令的 I 字段。)

▶ 26. [*32*] 本题是写一个读入卡片的程序。每台计算机为获得初始时进入计算机的信息以及正确地启动,都有它自己的特殊的"引导"问题。对于 MIX 来说,一张卡片的内容只能以字符代码的形式读入,因而含有装入程序本身的卡片也必须满足这一限制。并非所有可能的字节值都能从一张卡片输入,而且从卡片读入的每一个字都是正的。

MIX 还有一个我们在正文中未及说明的特性,这就是有一个"GO 按钮",当 MIX 的内存中含有任意信息时,它用来从杂乱中使计算机启动。当计算机操作员按下这一按钮时,发生下列的动作:

1) 一张卡片被读入到单元 0000 ~ 0015 中,这实际上等价于指令"IN 0(16)"。

2) 当这一卡片已被完全读入而卡片输入机不再忙碌时,就转移到单元 0000。J 寄存器也被置成零。

3) 机器现在开始执行已从卡片读入的这个程序。

注:没有卡片输入机的 MIX 计算机有附属于另一个输入设备的 GO 按钮。但在这个问题中我们将假定有卡片输入机,即设备 16。

所写的装入程序必须满足以下条件:

i) 输入卡片应以装入程序开始,跟随它的是包含要被装入的数的信息卡片,紧接是关掉装入程序的"转移卡片",并转到程序的开始处。装入程序应可存到两张卡片上。

ii) 信息卡片有下列格式:

1 ~ 5 列,为装入程序所忽略。

6 列,在这张卡片上要装入的连续的字的个数(这是 1 与 7 之间(含)的一个数)。

7 ~ 10 列,字 1 的内存单元,它总是大于 100(使得它不与装入程序相重叠)。

11 ~ 20 列,字 1。

21 ~ 30 列,字 2(如果列 6≥2)。

……

71 ~ 80 列,字 7(如果列 6 = 7)。

字 1,2,…的内容总是数值地穿孔为十进制数。如果一个字应该是负的,负号("11 穿孔")就被重叠穿孔到最低有效位,例如 20 列上。假定这使得字符代码输入是 10,11,12,…,19 而不是 30,31,32,…,39。例如,在 1 ~ 40 列上穿孔

ABCDE31000012345678900000000001000000010$\bar{0}$

的一张卡片引起下列数据被装入:

1000:　+0123456789;　1001:　+0000000001;　1002:　-0000000100

iii) 转移卡片在 1~10 列上有格式 TRANS0nnnn，其中 nnnn 是执行应当开始的位置。

iv) 装入程序应对所有字节大小都能进行工作，而无须对穿有装入程序的卡片作任何改动。卡片不应含有对应于字节 20,21,48,49,50,…(就是字符：Σ，Π，=，$，<，…)的任何字符，因为所有卡片输入机都不能输入这些字符。特别是，不能使用 ENT，INC 和 CMP，因为它们都不能穿到卡片上。

1.3.2 MIX 汇编语言

利用符号语言可以相当可观地简化 MIX 程序的阅读和编写工作，从而也减轻程序员对繁琐的书写细节的烦恼。这些繁琐的细节，常常导致不必要的错误。语言 MIXAL (“MIX Assembly Language”，MIX 汇编语言)，是对上节指令记号的扩充。它的主要特征是随意地使用字母名称来代表数，并随意地使用一个单元字段把名称同内存单元加以关联。

如果我们先考虑一个简单的例子，则 MIXAL 就很容易理解了。下列代码是一个较大的程序一部分；它是按照算法 1.2.10M，求 n 个元素 $X[1],\cdots,X[n]$ 的极大值的一个子程序。

程序 M(求极大值)　寄存器赋值：rA ≡ m，rI1 ≡ n，rI2 ≡ j，rI3 ≡ k，X[i] ≡ CONTENTS(X + i)。

汇编指令						行号	LOC	OP	ADDRESS	次数	注解
						01	X	EQU	1000		
						02		ORIG	3000		
3000:	+	3009	0	2	32	03	MAXIMUM	STJ	EXIT	1	子程序链接
3001:	+	0	1	2	51	04	INIT	ENT3	0,1	1	M1. 初始化。$k \leftarrow n$
3002:	+	3005	0	0	39	05		JMP	CHANGEM	1	$j \leftarrow n, m \leftarrow X[n], k \leftarrow n-1$
3003:	+	1000	3	5	56	06	LOOP	CMPA	X,3	$n-1$	M3. 比较
3004:	+	3007	0	7	39	07		JGE	*+3	$n-1$	如果 $m > X[k]$，转 M5
3005:	+	0	3	2	50	08	CHANGEM	ENT2	0,3	$A+1$	M4. 改变 m。$j \leftarrow k$
3006:	+	1000	3	5	08	09		LDA	X,3	$A+1$	$m \leftarrow X[k]$
3007:	+	1	0	1	51	10		DEC3	1	n	M5. k 减 1
3008:	+	3003	0	2	43	11		J3P	LOOP	n	M2. 全部试过了? 若 $k>0$，转 M3
3009:	+	3009	0	0	39	12	EXIT	JMP	*	1	返回主程序 ▌

这个程序是同时说明如下内容的例子：

a) 以“LOC”(单元)，“OP”(操作码)以及“ADDRESS”(地址)为标题的列是最主要的，它们包含以 MIXAL 符号机器语言写成的一个程序，以下我们将要说明这个程序的细节。

b)以“汇编指令”为标题的列表示对应于 MIXAL 程序的真实的机器语言。我们已经把 MIXAL 设计成使得任何 MIXAL 程序都能容易地翻译成数值机器语言,翻译通常是通过叫做汇编程序的另一个计算机程序进行的。于是,在 MIXAL 之下程序员可以去做他们的所有机器语言程序设计工作,而不必操心手工确定等价的数值代码。实际上本书中所有的 MIX 程序都是以 MIXAL 写成的。

c)以“行号”为标题的列并非 MIXAL 程序的基本部分,它仅被包括在本书中的 MIXAL 例子中,使得我们能容易地查阅程序的各部分。

d)以“注解”为标题的列给出关于程序的说明性信息,而且它同算法 1.2.10M 的各个步骤互相参照。读者应该把该算法同上边的程序加以对照。注意,在誊写成 MIX 代码时使用了一个小的“程序员的自由(license)”;例如,步骤 M2 已被放在后边。在程序 M 开始处指出的“寄存器赋值”示出 MIX 的什么元素对应于程序中的变量。

e)以“次数”为标题的列在本书中将要研究的许多 MIX 程序中是有启发性的。它代表了一个侧面(profile),即在该程序执行的过程中,该行上的指令将被执行的次数。因此行 06 将被执行 $n-1$ 次,等等。由这个信息,我们能够确定为实现这个子程序所需要的时间长度,它是$(5+5n+3A)u$,其中 A 是在 1.2.10 小节中已仔细分析过的量。

现在我们讨论程序 M 的 MIXAL 部分,行 01

```
X EQU 1000
```

指出符号 X 是同数 1000 等价的。这条指令的效果可以在行 06 上看出,其中指令“CMPA X,3”的数值等价表现为

+	1000	3	5	56

即“CMPA 1000,3”。

行 02 指出,后续诸行的单元应顺序地选择,并从 3000 开始。因此,出现在行 03 的 LOC 字段的符号 MAXIMUM 变成等价于数 3000,INIT 等价于 3001,LOOP 等价于 3003,等等。

在行 03 到 12 的 OP 字段包含 MIX 指令的符号名:STJ,ENT3,等等。但是出现在行 01 和行 02 上的 OP 列的符号名 EQU 和 ORIG,是稍微不同的;EQU 和 ORIG 称做伪操作,因为它们是 MIXAL 的操作符但不是 MIX 的操作符。伪操作提供了关于一个符号程序的专门信息,而不是程序本身的指令。因此,行

```
X EQU 1000
```

仅仅谈及程序 M,它并不表明当程序运行时有哪一个变量被置成等于 1000。注意,行 01 和 02 没有汇编出指令来。

行 03 是一条“存储 J”指令,它把寄存器 J 的内容存入单元 EXIT 的(0:2)字段中。换言之,它把 rJ 存储到在行 12 指令的地址部分。

如同较早时提到过的那样,程序 M 是一个较大的程序的一个部分;在别处,序列

```
ENT1 100
JMP MAXIMUM
STA MAX
```

将,例如,把 n 设置为 100,再转移到程序 M 来。程序 M 然后将求出 $X[1],\cdots,X[100]$的最大值,并且返回到指令“STA MAX”,在 rA 中保留它的极大值,而在 rI2 中保留它的位置 j。(参见习题 3。)

行 05 把控制转移到行 08。行 04,05,06 不需要作进一步的说明。行 07 引进一个新的记号:一个星号(读作“自己”)指行出现的位置;“ * + 3”(“自己加 3”)因此指从当前行往下三行。由于行 07 是对应于单元 3004 的一条指令,“ * + 3”出现在那里便指单元 3007。

剩下的符号代码是自明的。注意在行 12 上再一个星号的出现(参见习题 2)。

我们的下一个例子引入汇编语言更多一些的特性。这个例子的任务是计算和打印头 500 个素数的一个表,且分成 10 列,每列 50 个数。在行式打印机上,这个表应呈现如下形式:

```
FIRST  FIVE  HUNDRED  PRIMES
0002   0233   0547   0877   1229   1597   1993   2371   2749   3187
0003   0239   0557   0881   1231   1601   1997   2377   2753   3191
0005   0241   0563   0883   1237   1607   1999   2381   2767   3203
0007   0251   0569   0887   1249   1609   2003   2383   2777   3209
0011   0257   0571   0907   1259   1613   2011   2389   2789   3217
 ⋮                                                 ⋮
0229   0541   0863   1223   1583   1987   2357   2741   3181   3571
```

我们将使用下列方法:

算法 P(打印 500 个素数的表)　本算法分为两个不同的部分:步骤 P1 ~ P8 准备一个 500 个素数的内部表,而步骤 P9 ~ P11 是以上表的形式打印出答案。算法的后边部分用了两个“缓冲区”,在其中形成行的映像;在打印一个缓冲区内容的同时,往另一个缓冲区送入数。

P1. [开始造表] 置 PRIME[1]←2,N←3,J←1。(在这个算法中,N 取遍作为素数的候选者的奇数;J 记住迄今为止已求出了多少个素数。)

P2. [N 是素数] 置 J←J + 1,PRIME[J]←N。

P3. [找到 500 个了?] 如果 J = 500,则转 P9。

P4. [增加 N] 置 N←N + 2。

P5. [K←2] 置 K←2。(PRIME[K]将取遍 N 的所有素因子。)

P6. [PRIME[K] \ N?] 以 PRIME[K]除 N;设 Q 为商而 R 为余数。如果 R = 0(因此 N 不是素数),则转 P4。

P7. [PRIME[K]足够大了吗?] 如果 Q≤PRIME[K],转 P2。(在这样的情况下,N 必定为素数;这个事实的证明是有趣的,而且有一点不同寻常——参见习题 6。)

P8. [K 加 1] K 加 1,并转 P6。

P9. [打印标题] 现在我们已做好打印表的准备,把打印机推进到下一页,置 BUFFER[0]为标题行并打印此行。置 B←1,M←1。

P10. [设置行] 以适当格式把 PRIME[M],PRIME[50 + M],…,PRIME[450 + M]放进 BUFFER[B]中。

P11. [打印行] 打印 BUFFER[B];置 B←1 - B (因此转到另一个缓冲区);且 M 加 1。若 M≤50,返回 P10;否则算法终止。 ▌

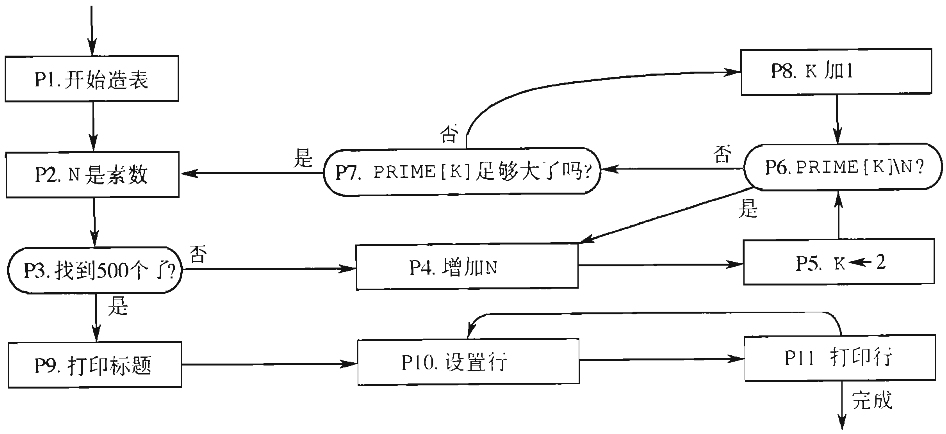


图 14 算法 P

程序 P(打印 500 个素数的表) 这个程序已故意地以稍微笨拙的风格来编写,以便可以在一个程序中解说 MIXAL 的大多数特性。rI1 ≡ J - 500; rI2 ≡ N; rI3 ≡ K; rI4 指示 B;rI5 是 M 加上 50 的倍数。

```
* EXAMPLE PROGPAM ... TABLE OF PRIMES
*
L        EQU   500          要求的素数个数
PRINTER  EQU   18           行式打印机设备号
PRIME    EQU   -1           素数表的存储区域
BUF0     EQU   2000         BUFFER[0]的存储区域
BUF1     EQU   BUF0+25      BUFFER[1]的存储区域
         ORIG  3000
START    IOC   0(PRINTER)   跳到新的页
         LD1   =1-L=        P1. 开始造表。J←1
         LD2   =3=            N←3
2H       INC1  1            P2. N是素数。J←J+1
         ST2   PRIME+L,1      PRIME[J]←N
         J1Z   2F           P3. 找到500个了吗?
4H       INC2  2            P4. 增加N
         ENT3  2            P5. K←2
```

```
6H      ENTA  0                  P6. PRIME[K] \ N?
        ENTX  0,2                  rAX←N
        DIV   PRIME,3              rA←Q,rX←R
        JXZ   4B                   如果 R = 0 转 P4
        CMPA  PRIME,3            P7. PRIME[K]足够大了吗?
        INC3  1                  P8. K加1
        JG    6B                   如果 Q > PRIME[K]转 P6
        JMP   2B                   否则 N 是素数
2H      OUT   TITLE(PRINTER)     P9. 打印标题
        ENT4  BUF1 + 10            置 B←1
        ENT5  - 50                 置 M←0
2H      INC5  L + 1                M 增加 1
4H      LDA   PRIME,5            P10. 设置行。(自右至左)
        CHAR                       把 PRIME[M]转换成十进数
        STX   0,4 (1:4)
        DEC4  1
        DEC5  50                   (rI5 减 50 直到它变成非正的数为止)
        J5P   4B
        OUT   0,4(PRINTER)       P11. 打印行
        LD4   24,4                 转换缓冲区
        J5N   2B                   如果 rI5 = 0,任务完成
        HLT
* INITIAL CONTENTS OF TABLES AND BUFFERS
        ORIG  PRIME + 1
        CON   2                  头一个素数为 2
        ORIG  BUF0 - 5
TITLE   ALF   FIRST              标题行的字母信息
        ALF    FIVE
        ALF    HUND
        ALF   RED P
        ALF   RIMES
        ORIG  BUF0 + 24
        CON   BUF1 + 10          每一个缓冲区引用另一个
        ORIG  BUF1 + 24
        CON   BUF0 + 10
        END   START              程序结束。 ▌
```

下面是关于这一程序值得注意的几点：

1. 行 01,02 和 39 以一个星号开始:表示这是仅起说明作用的“注释行”,它对被汇编的程序没有实际的影响。

2. 如同在程序 M 中那样,行 03 中的 EQU 设置一个符号的等价值;在此情况下,与 L 等价的值是 500。(在行 10~24 的程序中,L 表示要被计算的素数的个数。)注意在行 05 中符号 PRIME 获得的一个负的等价值;一个符号的等价值可以是任何带符号的五个字节的数。在行 07 中 BUF1 的等价值被计算作为 BUF0+25,即 2025。MIXAL 提供了有限数量的算术运算;另外的例子可在行 13 中见到,其中 PRIME+L 的值(在现在的情况下,它的值是 449)是通过汇编程序计算的。

3. 在行 25 和 35 中符号 PRINTER 已被用于 F 部分中。F 部分总是用圆括号括住,既可以是数值的,也可以是符号的,就如同 ADDRESS 字段的其它部分一样。行 31 使用冒号说明了部分字段的描述“(1:4)”。

4. MIXAL 提供了好几种方法来说明非指令字。行 41 使用伪操作码 CON 来说明一个普通常数“2”;行 41 的结果是汇编字

+	2

行 49 示出一个稍微更复杂的常数“BUF1+10”,它汇编成字

+	2035

一个常数可以被包括在两个等号当中,在此情况下我们称之为一个*文字常数*(参见行 10 和行 11)。汇编程序自动地建立文字常数的内部名并插入“CON”行。例如,程序 P 的行 10 和行 11 被有效地改变成

```
10    LD1    con1
11    LD2    con2
```

而后,在程序的末尾处,在行 51 和 52 之间,行

```
51a    con1    CON  1-L
51b    con2    CON  3
```

被有效地插入作为汇编过程的一部分。行 51a 将汇编出字

−	499

使用文字常数肯定是很方便的,因为它意味着程序员不需要去对平凡的常数杜撰一个符号名,也不需要记住在每个程序的末尾插入常数。程序员可以把自己的思想集中到中心问题上,而且无须去为这样的程序细节操心。(然而,在程序 P 中的文字常数并不是特别好的例子,因为如果我们以更为有效的命令“ENT1 1-L”和“ENT2 3”来代替行 10 和 11,我们就将有一个稍微更好的程序。)

5. 一个好的汇编语言应当模拟一个程序员考虑机器程序的方式。这个哲理的一个例子就是使用文字常数,如同我们刚刚提到的那样;另一个例子是使用“*”,在程序 M 中我们已说明过了。第三个例子是如符号 2H 这样的*局部符号*,它出现在行 12,25 和 28 的单元字段处。

局部符号是特殊的符号,它的等价值可以随意地多次*重新定义*。像 PRIME 这样一

个全局符号在整个程序中自始至终仅有一个意义，而如果它出现在多于一行的单元字段处，则汇编程序就将指出一个错误来。但局部符号却有一个不同的性质。例如：我们在单元字段中写 2H（"这里 2"），以及在一个 MIXAL 行的地址字段中写 2F（"向前 2"）和 2B（"向后 2"）*：

2B　指上一个最近的单元 2H

2F　指下一个最近的单元 2H

因此，行 14 中的"2F"是指的行 25；在行 24 中的"2B"向后指行 12；而在行 37 中的"2B"指的是行 28。2F 或 2B 的地址绝不指它自己的行；例如 MIXAL 代码的三行

```
2H  EQU   10
2H  MOVE  2F(2B)
2H  EQU   2B-3
```

实质上等价于这样一行

```
MOVE  *-3(10)
```

符号 2F 和 2B 不得应用于单元字段中；符号 2H 不得应用于地址字段中。共有十个局部符号，它们可以通过在这些例子中用从 0 到 9 的数字代替"2"得到。

局部符号的思想是由 M. E. Conway 于 1958 年，在关于 UNIVAC I 的一个汇编程序中引进的。局部符号使得当程序员要去访问前后几行之外的一个指令时，不必去对每一个地址都选择符号名。临近的单元通常都没有适当的名字，因此程序员就倾向于使用像 X1，X2，X3 等无意义的符号，因而有潜在的重复使用的危险。因此在一个汇编语言中局部符号是十分有用和自然的。

6. 行 30 和 38 的地址部分是空白的，这意味着被汇编出的地址是 0。同样地，我们也可以使行 17 的地址也保持空白，但如果没有这冗余的 0 将使程序不大容易阅读。

7. 行 43～47 使用"ALF"操作，它以 MIX 的字符代码形式建立一个五字节的常数，例如，行 45 引起字

+	00	08	24	15	04

被汇编，表示"␣HUND"——程序 P 的输出中标题行的一部分。

*所有在 MIXAL 程序中未予说明其内容的单元，一般都置成*0（但由装入程序所使用的单元除外，这些单元通常是 3700～3999）。因此，在行 47 之后，不需要把标题行的其它字置成空白。

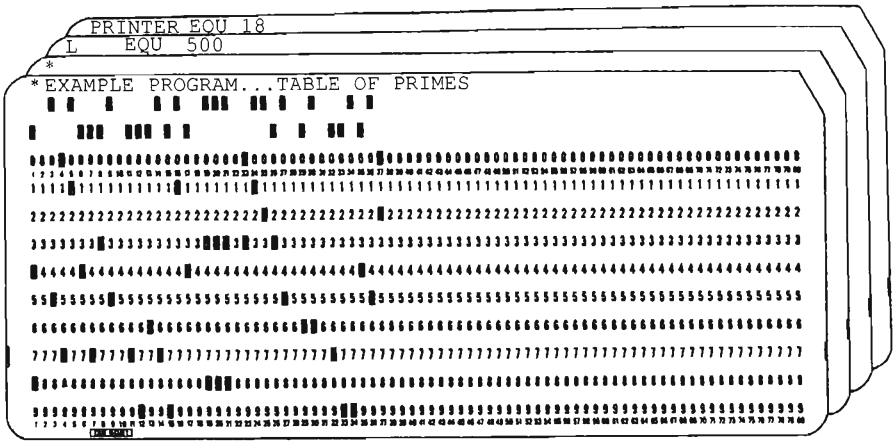
8. 算术运算可以和 ORIG 一起使用：参见行 40，42 和 48。

9. 一个完整的 MIXAL 程序的最后一行总有 OP 代码"END"。在这一行上的地址是一旦程序装入内存之后，程序的开始单元。

10. 作为对于程序 P 的最后的注释，我们可以观察到，指令已被组织成使得变址寄存器是向着零来计数，因此只要可能就对它测试是否为零。例如，在 rI1 中保存的是 J－500，而不是 J。尽管或许有点儿棘手，但行 26～34 是特别值得注意的。

---

* 这里 H，F 和 B 与英文单词 here，forward 和 backward 对应。——译者注




```
* EXAMPLE PROGRAM···TABLE OF PRIMES
*
L EQU 500
PRINTER EQU 18
PRIME EQU -1
BUF0 EQU 2000
BUF1 EQU BUF0 +25
 ORIG 3000
START IOC 0(PRINTER)
 LD1 =1-L=
```

图 15　在卡片上穿孔的或者在一个终端上打入的程序 P 开头的一些行

当程序 P 真正运行时所发现的一些统计数字，指出来或许也是有益的。在行 19 上的除法指令被执行了 9538 次；行 10 ~ 24 执行的时间为 182144u。

如同图 15 所示，MIXAL 程序可以穿孔到卡片上或在一台计算机终端上敲入。在穿孔卡片的情况下使用下列格式：

列　1 ~ 10　LOC(单元)字段
列　12 ~ 15　OP 字段
列　17 ~ 80　ADDRESS 字段和任选的注释
列　11,16　空白

然而，如果列 1 包含一个星号，则整张卡片就被当做一个注释。ADDRESS 字段以列 16 之后的头一个空白列结束；任何说明性的信息都可以被穿孔在这个头一个空白格的右边而对汇编程序没有影响。(例外：当 OP 字段是 ALF 时，注释总是从列 22 开始。)

当输入来自一个终端时，使用限制较少的格式：LOC 字段以头一个空白结束，而 OP 和 ADDRESS 字段(如果存在的话)，以一个非空字符开始而且继续到下一个空格；然而，特殊的 OP 代码 ALF 或者后边跟随两个空格及五个字母数字字符，或者后边跟随一个空格及五个字母数字字符，其中的头一个是非空的。每行的剩余部分包含任选的注释。

MIX 汇编程序接收以这个方式准备的输入文件并以可装入的形式把它们转换成机器语言程序。在有利的环境下读者将要接触一个 MIX 汇编程序和 MIX 模拟器，通过它们本书中的各种习题都可求解。

现在我们已经看到在 MIXAL 中可以做些什么。我们以更仔细地描述一些规则，特别是我们将观察在 MIXAL 中什么是不允许做的，来结束这一小节。以下用相当少的规则定义这个语言。

1. 一个符号是一至十个字母和或数字的串，而且至少要有一个字母。例如：PRIME，TEMP，20BY20。特殊符号 dH，dF 以及 dB，其中 d 是一位数字，将由于这个定义的目的，而按照早先所描述的"局部符号"的约定，由其它惟一的符号所代替。

2. 一个数是一至十个数字的串。例如：00052。

3. 在一个 MIXAL 程序中一个符号的每一个出现或说是一个"已定义的符号"，或说是"待引用"。一个已定义的符号是已经在这个 MIXAL 程序的前边行的 LOC 字段中出现了的符号。一个待引用的符号是还未以这种方式定义的符号。

4. 一个原子表达式是

a)一个数，或

b)一个已定义的符号(表示该符号的数值的等价值，参见规则 13)，或

c)一个星号(表示⊛的值；参见规则 10 和 11)。

5. 一个表达式是

a)一个原子表达式，或

b)一个加号或减号后边跟着一个原子表达式，或

c)一个表达式，后边跟随着一个二元操作符，再跟着一个原子表达式。

六个允许的二元操作是 +，-，*，/，//以及:。它们对于数值的 MIX 字定义如下：

```
C = A + B     LDA  AA;    ADD BB;     STA CC
C = A - B     LDA  AA;    SUB BB;     STA CC
C = A * B     LDA  AA;    MUL BB;     STX CC
C = A/B       LDA  AA;    SRAX 5;     DIV BB;   STA CC
C = A//B      LDA  AA;    ENTX 0;     DIV BB;   STA CC
C = A:B       LDA  AA;    MUL = 8 =;  SLAX 5;   ADD BB; STA CC
```

这里 AA，BB 和 CC 表示包含 A，B 和 C 相应值的单元。在一个表达式内的操作是从左至右地进行的，例如：

-1+5	等于4
-1+5*20/6	等于 4*20/6 等于 80/6 等于13(从左到右进行)
1//3	等于一个 MIX 字，其值近似于 $b^5/3$，其中 b 是字节大小；即表示分数 1/3 的一个字且有一个假定的小数点在左边
1:3	等于11(通常用在部分字段的说明)
*-3	等于⊛-3
***	等于⊛乘⊛

6. 一个 A 部分(它用来描述一条 MIX 指令的地址字段)是

a)空白(表示值 0),或

b)一个表达式,或

c)一个待引用(表示与这符号等价的最后的值;参见规则 13),或

d)一个文字常数(表示对一个内部建立的符号的引用;参见规则 12)。

7. 一个变址部分(它用来描述一条 MIX 指令的变址字段)是

a)空白(表示值 0),或

b)一个逗号,后边接上一个表达式(表示该表达式的值)。

8. 一个 F 部分,它用来描述一条 MIX 指令的 F 字段。它是

a)空白(根据表 1.3.1-1 所示的 OP 字段,表示正常的 F 设置),或

b)一个左圆括号,后边跟着一个表达式,后边再跟着一个右圆括号(表示一个表达式的值)。

9. 一个 W 值(它用来描述 MIX 的一个全字长的常数),它是

a)一个表达式,后边接上一个 F 部分(在此情况下,一个空白的 F 部分表示(0:5)),或

b)一个 W 值后边跟着一个逗号,后边再跟着形同 a)的 W 值。

一个 W 值表示如下确定的数值 MIX 字的值:令 W 值有"$E_1(F_1)$, $E_2(F_2)$, …, $E_n(F_n)$"的形式,其中 $n \geq 1$,诸 E 是表达式,诸 F 是字段。如果以下假设的程序被执行,则所求的结果是将出现在内存单元 WVAL 中的最后的值:

STZ WVAL; LDA C_1; STA WVAL(F_1); …; LDA C_n; STA WVAL(F_n)

这里 $C_1, \ldots, C_n$ 表示包含表达式 $E_1, \ldots, E_n$ 的值的单元。每一个 F_i 必须有 $8L_i + R_i$ 的形式,其中 $0 \leq L_i \leq R_i \leq 5$。例子:

1	是字	+		1
1, -1000(0:2)	是字	-	1000	1
-1000(0:2),1	是字	+		1

10. 汇编的过程利用由⊛(称为单元计数器)所表示的一个值,它开始时为 0。⊛的值总应是可以装入两个字节中的一个非负数。当一行的单元字段不是空白时,它必须包含以前还未定义的一个符号。该符号的等价值则被定义为⊛的当前值。

11. 如同规则 10 所描述的那样处理了 LOC 字段后,汇编过程取决于 OP 字段的值。OP 有六种可能的值:

a) OP 是一个 MIX 符号操作符(参见上一节末尾处的表 1)。该表定义了每个 MIX 操作符的正常的 C 和 F 值。在此情况下,ADDRESS 应是一个 A 部分(规则 6),后边跟着一个变址部分(规则 7),后边再跟着一个 F 部分(规则 8)。因此我们得到四个值:C,F,A 和 I,其效果是把由序列"LDA C; STA WORD; LDA F; STA WORD(4:4); LDA I; STA WORD(3:3); LDA A; STA WORD (0:2)"所确定的字汇编装入由⊛所指定的单元中,并使⊛加 1。

b) OP 是“EQU”。其 ADDRESS 应是一个 W 值(参见规则 9)。如果 LOC 字段是非空的,则出现在那儿的符号的等价值被置成等于在 ADDRESS 中确定的值。这个规则优先于规则 10,⊛的值不变。(作为一个并不平常的例子,考虑行

BYTESIZE EQU 1(4:4)

它允许程序员有一个其值依赖于字节大小的符号。只要得到的程序对于每个可能的字节是有意义的,这个情况就是可接受的。)

c) OP 是“ORIG”。ADDRESS 应是一个 W 值(参见规则 9);单元计数器⊛被置成这个值。(注意由于规则 10,出现在一个 ORIG 行的 LOC 字段中的一个符号在⊛的值改变之前获得⊛的值作为它的等价值。例如,

TABLE　ORIG　*+100

置 TABLE 的等价值为 100 个单元的头一个。

d) OP 是“CON”。ADDRESS 应是一个 W 值;其效果是以这个值把一个字汇编装入由⊛所确定的单元中,并把⊛加 1。

e)OP 是“ALF”。其效果是汇编由地址字段的头五个字符组成的字符代码的字,其余特性就像 CON 那样。

f)OP 是“END”。ADDRESS 应是一个 W 值,它在这个指令的(4:5)字段中,确定程序开始处的指令的单元。END 行标记一个 MIXAL 程序的结束。汇编程序以任意顺序把附加的行插入在 END 行之前,它们对应于所有未定义的符号和文字常数(参见规则 12 和 13)。因此在 END 行的 LOC 字段中的一个符号将表示紧跟被插入的字后面的头一个单元。

12. 文字常数:小于 10 个字符长的一个 W 值,可以括在两个 = 号之间并且用作一个未来的引用。其效果是内部地建立一个新符号和插入定义该符号的一个 CON 行于 END 行的紧前边(参见程序 P 后边的注释 4)。

13. 每一个符号有一个且只有一个等价值;这是一个完整字的 MIX 数,通常它是按照规则 10 或规则 11b)由在 LOC 中的符号的出现确定。如果符号在 LOC 中不出现,则在 END 行之前有效地插入一个新行,并且有 OP =“CON”,ADDRESS =“0”以及符号名于 LOC 中。

注:上边的规则最重要的结果是对未来引用的限制。在上一个行的 LOC 字段中尚未定义的一个符号不能使用,除非作为一个指令的 A 部分。特别是,它不可以被使用于(a)同算术运算相关联,或者(b)EQU,ORIG 或 CON 的 ADDRESS 字段中。例如,

```
        LDA  2F+1
以及
        CON  3F
```

都不合法。为了允许对程序更有效的汇编而强加了这个限制,而且在编写这套书时所获得的经验已经证明,它是一个很少造成很大差别的温和的限制。

实际上 MIX 有两个用于低级程序设计的符号语言:MIXAL*,它被设计成便于由一

* 作者于 1971 年惊讶地得知,MIXAL 也是南斯拉夫的一种洗涤剂的名称,这种洗涤剂是为自动洗衣机而生产的。——原注

个非常简单的汇编程序进行一遍翻译的面向机器的语言;以及 PL/MIX,它更适合于反映数据和控制结构,而且它看起来很像是 MIXAL 程序的注释字段。我们将在第 10 章描述 PL/MIX。

习 题——第一组

1. [*00*] 正文说明,“X EQU 1000”并不汇编出设置一个变量的值的任何指令。假设你正在编写一个 MIX 程序,在这个程序中你想把包含于某个内存单元(其符号名为 X)的值置成 1000。你怎样以 MIXAL 写出这个程序?

▶ 2. [*10*] 程序 M 的行 12 指出“JMP *”。这里 * 表示该行的单元。为什么这个程序不陷入一个无穷循环,无穷地重复这一指令?

▶ 3. [*23*] 如果把下列程序同程序 M 连接在一起使用,它的效果是什么?

```
START   IN    X+1(0)
        JBUS  *(0)
        ENT1  100
1H      JMP   MAXIMUM
        LDX   X,1
        STA   X,1
        STX   X,2
        DEC1  1
        J1P   1B
        OUT   X+1(1)
        HLT
        END   START   ▌
```

▶ 4. [*25*] 用手工把程序 P 汇编出来。(它所花的时间不会如你想像的那么长。)对应于该符号程序,内存的真正的数值内容是什么?

5. [*11*] 为什么程序 P 不需要一条 JBUS 指令来确定什么时候行式打印机就绪?

6. [*HM20*] (a)证明如果 n 不是素数,n 有满足 $1 < d \leq \sqrt{n}$ 的一个因子 d。(b)使用这一事实,说明,算法 P 的步骤 P7 中的测试可证 N 是一个素数。

7. [*10*] (a)程序 P 的 34 行的“4B”的含义是什么? (b)如果把 15 行的单元改成“2H”而把 20 行的地址改成为“2B”,如果有效果的话,引起的效果是什么?

▶ 8. [*24*] 下列程序做什么?(不要在一台计算机上运行它,而是手工把它弄清楚!)

```
* MYSTERY PROGRAM
BUF  ORIG  *+3000
1H   ENT1  1
     ENT2  0
     LDX   4F
2H   ENT3  0,1
3H   STZ   BUF,2
     INC2  1
```

```
        DEC3  1
        J3P   3B
        STX   BUF,2
        INC2  1
        INC1  1
        CMP1  =75=
        JL    2B
        ENN2  2400
        OUT   BUF+2400,2(18)
        INC2  24
        J2N   *-2
        HLT
4H      ALF   AAAAA
        END   1B  ▌
```

习　题——第二组

下列习题都是短的程序设计问题，它们代表了典型的计算机应用，并且覆盖了广泛的技术。为了获得使用 MIX 的一些经验，也为了能对基本的程序设计技巧进行很好的复习，我们鼓励每个读者都选择其中的一些题目来做一做。如果愿意的话，也可以在阅读第 1 章的剩余部分时同时来做这些题。

以下的一览表指出所涉及的程序设计技术的类型：

使用多重决策的开关表：习题 9，13 和 23。

对二维数组使用变址寄存器：习题 10，21 和 23。

字符的解包：习题 13 和 23。

整数和舍入小数的算术运算：习题 14，16 和 18。

使用子程序：习题 14 和 20。

输入缓冲：习题 13。

输出缓冲：习题 21 和 23。

列表处理：习题 22。

实时控制：习题 20。

图形显示：习题 23。

每当本书中的一个习题指出“试写一个 MIX 程序”或“试写出一个 MIX 子程序”时，你只需对于所要求的问题写符号 MIXAL 代码就行了。这个代码本身将不是完整的，它将仅仅是（设想的）一个完整的程序的一个片段。如果数据是从外部提供的，则在这个程序段中就不必进行输入或输出；需要写的仅仅是 MIXAL 行的 LOC，OP 以及 ADDRESS 字段，连同适当的注释。除非有明确要求，否则便不要求写数值的机器语言、行号以及“时间”列（参见程序 M），也将没有一个 END 行。

另一方面，如果一道习题说“试写出一个完整的 MIX 程序”，那它意味着，应当以 MIXAL 写出一个可执行的程序，特别应包括最后的 END 行。用于测试这样的完整程序的汇编程序和 MIX 模

拟程序可以广泛地获得。

▶ 9. [*25*] 单元 INST 中有一个疑似一条 MIX 指令的 MIX 字。试编写一个 MIX 程序，如果按照表 1.3.1-1，该字含有一个有效的 C 字段，有效的 ± AA 字段，有效的 I 字段，以及有效的 F 字段的话，它转到单元 GOOD 去。否则，你的程序应转到单元 BAD 去。记住，对于一个有效的 F 字段的测试依赖于 C 字段。例如，如果 C = 7(MOVE)，则任何 F 字段都是可接受的。但如果 C = 8 (LDA)，则 F 字段必须有 8L + R 的形式，其中 $0\le L\le R\le 5$。除非 C 指明要求一个内存地址和 I = 0 的指令，此时 ± AA 不是一个有效的内存地址，否则 ± AA 字段应被认为是正确的内存地址。

注：没有经验的程序员倾向于通过编写对 C 字段的一个长的测试系列来解决类似于上述的问题。例如，"LDA C;JAZ 1F;DECA 5;JAN 2F;JAZ 3F;DECA 2;JAN 4F;…"这不是一个好的做法！进行多路判断的最好方法是编制一张包括所要求逻辑信息的辅助表。例如，如果有一个含 64 个项的表，我们可以写"LD1 C;LD1 TABLE,1;JMP 0,1"——由此非常快速地转移到所需的程序。在这样的表中也可保存其它有用的信息。对于我们当前的问题造表方法只不过使程序变得稍长些(包括有表在内)，但却大大地增加了它的速度和灵活性。

▶ 10. [*31*] 假设我们有一个存于内存中的 9 × 8 矩阵

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{18} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{28} \\ \vdots & & & & \vdots \\ a_{91} & a_{92} & a_{93} & \cdots & a_{98} \end{pmatrix}$$

使得 a_{ij} 是在单元 $1000+8i+j$ 中。因此在内存中这个矩阵以如下形式出现

$$\begin{pmatrix} (1009) & (1010) & (1011) & \cdots & (1016) \\ (1017) & (1018) & (1019) & \cdots & (1024) \\ \vdots & & & & \vdots \\ (1073) & (1074) & (1075) & \cdots & (1080) \end{pmatrix}$$

如果某个位置是它所在的行中的最小值和它所在的列中的最大值，则说这个矩阵有一个"鞍点"。用符号来表示，如果

$$a_{ij} = \min_{1\le k\le 8} a_{ik} = \max_{1\le k\le 9} a_{kj}$$

则 a_{ij} 是一个鞍点。试写出一个计算鞍点(如果至少有一个的话)或零(如果没有鞍点的话)的位置的 MIX 程序，而且在停机时上述的值存于 rI1 中。

11. [*M29*] 假定在上题中的 72 个元素都不相同而且假定所有 72! 种安排都是概率相同的，问出现一个鞍点的概率是多少？如果我们假定矩阵的元素是 0 和 1，而且所有 2^{72} 种这样的矩阵都是概率相同的，问相应的概率是多少？

12. [*HM42*] 对于习题 10 给出了两个解(见习题答案)，并且还提示了第三个解；但不清楚哪一个解是最好的。试利用习题 11 的每个假定，对这些算法进行分析，判定哪一个是较好的方法。

13. [*28*] 一位密码分析家要计算在某一密码中字母的出现频率。这个密码已经在纸带上被穿孔了，并且其末尾以一个星号来标出。试写出一个完整的 MIX 程序，它把这个纸带输入进来，统计直到头一个星号为止每个字符的频率，而后以下列形式打印出结果：

```
A  0010257
B  0000179
D  0794301
```

等等，每行一个字符。不计空格的个数，其计数为零的字符(像在上面的 C)不打印。为了提高效率，把输入加以“缓冲”：当把一个块输入到内存的一个区域时，你可以从另一个区域统计字符。你还可以假定，在输入带上还存在一个额外的块(在含有结束的星号那个块之后)。

▶ 14. [*31*] 下列算法，是意大利那不勒斯天文学家 Aloysius Lilius 和德国耶稣会数学家 Christopher Clavius 在 16 世纪末给出的。这个算法被大多数西方教会用来确定 1582 年之后任何一年的复活节星期日的日期。

算法 E(复活节的日期)　设 Y 是欲求复活节日期的年份。

E1. [黄金数] 置 $G \leftarrow (Y \bmod 19) + 1$。($G$ 是在 19 年默冬(Metonic)周期下这年的所谓“黄金数”。)

E2. [世纪] 置 $C \leftarrow \lfloor Y/100 \rfloor + 1$。(当 Y 不是 100 的倍数时，C 是世纪数；例如，1984 年处于 20 世纪中。)

E3. [校正] 置 $X \leftarrow \lfloor 3C/4 \rfloor - 12$，$Z \leftarrow \lfloor (8C+5)/25 \rfloor - 5$。(这里 X 是年份，例如 1900。其中跳过了闰年以便同太阳的运行同步；Z 是用来使复活节同月亮的轨道同步的一个特殊校正值。)

E4. [求星期日] 置 $D \leftarrow \lfloor 5Y/4 \rfloor - X - 10$。(3 月 $((-D) \bmod 7)$ 日实际上将是一个星期日。)

E5. [闰余] 置 $E \leftarrow (11G + 20 + Z - X) \bmod 30$。如果 $E = 25$，且黄金数 G 大于 11，或者如果 $E = 24$，则 E 加 1。(这个数 E 是闰余，它确定满月出现的时间。)

E6. [求满月] 置 $N \leftarrow 44 - E$。如果 $N < 21$，则置 $N \leftarrow N + 30$。(复活节应当是在 3 月 21 日起出现的头一个满月之后的头一个星期日。实际上，月亮轨道的摄动使这点并不严格地准确。但是这里我们关心的是“日历的月亮”，而不是真正的月亮，三月 N 日是日历的一个满月。)

E7. [进到星期日] 置 $N \leftarrow N + 7 - ((D + N) \bmod 7)$。

E8. [得出月份] 如果 $N > 31$，则日期是 4 月 $(N-31)$ 日 ($(N-31)$APRIL)，否则日期是 3 月 N 日 (N MARCH)。▌

假定年份小于 100000，试写出计算和打印给定年份的复活节日期的一个子程序。输出应有“*dd* MONTH, *yyyyy*”的形式，其中 *dd* 是日期而 *yyyyy* 是年份。试编写一个完整的 MIX 程序，它使用这个子程序来编制从 1950 年起直至 2000 年止的复活节日期的表。

15. [*M30*] 在编制上题的程序中，一个相当普遍的错误是没有认识到，在步骤 E5 中的量 $(11G + 20 + Z - X)$ 可能是负数；因此有时算出来的不是正的模 30 余数。(参见 *CACM* **5** (1962), 556。)例如，在 14250 年，我们将求出 $G = 1$，$X = 95$，$Z = 40$；所以如果我们有 $E = -24$ 而不是 $E = +6$，我们就将得到荒谬的答案“42 APRIL”。试写出一个完整的 MIX 程序，它找出将造成所算出的复活节日期出错的最早的年份。

16. [*31*] 在 1.2.7 小节我们证明了和数 $1 + \frac{1}{2} + \frac{1}{3} + \cdots$ 变成无穷大。但是如果通过一台计算机以有限的精度来计算这个和数，则在某种意义下，这个和实际上存在，因为后边的项逐渐地变成如此之小，以致我们如果逐项把它们加起来，它们对和数无所贡献。例如，假设我们通过舍入到头一位小数来进行计算，则我们有 $1 + 0.5 + 0.3 + 0.3 + 0.2 + 0.2 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 = 3.9$。

更准确地说，令 $r_n(x)$ 是舍入到小数点后 n 位的数 x；我们定义 $r_n(x) = \lfloor 10^n x + \frac{1}{2} \rfloor / 10^n$。然后我们希望求

$$S_n = r_n(1) + r_n\left(\frac{1}{2}\right) + r_n\left(\frac{1}{3}\right) + \cdots$$

我们知道 $S_1 = 3.9$，问题是要写出一个完整的 MIX 程序，它对于 $n = 2,3,4$ 和 5 计算和打印 S_n。

注：实现这个求和，有比一次加一个数地来加 $r_n(1/m)$，直到 $r_n(1/m)$ 变成 0 为止这一简单过程要快得多的方法。例如，对于从 66667 直到 200000 为止的所有 m 值，我们都有 $r_5(1/m) = 0.00001$；避免计算 $1/m$ 达 133334 次是明智的！倒是应该使用沿着下列诸行的一个算法：

A. 以 $m_h = 1, S = 1$ 开始。

B. 置 $m_e = m_h + 1$ 并计算 $r_n(1/m_e) = r$。

C. 求 m_h，即使得 $r_n(1/m) = r$ 的最大的 m。

D. 把 $(m_h - m_e + 1)r$ 加到 S 上并返回步骤 B。

17. [*HM30*] 使用上题的记号，证明或者反驳公式

$$\lim_{n\to\infty}(S_{n+1} - S_n) = \ln 10$$

18. [*25*] 分母小于等于 n 的 0 和 1 之间所有既约分数的递增序列称做"n 阶法里(Farey)级数"。例如，7 阶法里级数是

$$\frac{0}{1},\frac{1}{7},\frac{1}{6},\frac{1}{5},\frac{1}{4},\frac{2}{7},\frac{1}{3},\frac{2}{5},\frac{3}{7},\frac{1}{2},\frac{4}{7},\frac{3}{5},\frac{2}{3},\frac{5}{7},\frac{3}{4},\frac{4}{5},\frac{5}{6},\frac{6}{7},\frac{1}{1}$$

如果我们以 $x_0/y_0, x_1/y_1, x_2/y_2, \cdots$ 来表示这个级数，习题 19 证明

$$x_0 = 0,\quad y_0 = 1;\quad x_1 = 1,\quad y_1 = n$$
$$x_{k+2} = \lfloor (y_k + n)/y_{k+1} \rfloor x_{k+1} - x_k$$
$$y_{k+2} = \lfloor (y_k + n)/y_{k+1} \rfloor y_{k+1} - y_k$$

试写出一个 MIX 子程序，它通过把 x_k 和 y_k 的值分别存放于单元 X + k 和 Y + k 中来计算 n 阶法里级数。(在这个级数中项的总数大约是 $3n^2/\pi^2$，所以你可以假定 n 是相当小的。)

19. [*M30*] (a)证明由上题的递推式定义的数 x_k 和 y_k 满足关系 $x_{k+1}y_k - x_ky_{k+1} = 1$。(b)利用(a)中证明了的事实，证明分数 x_k/y_k 确实是 n 阶法里级数。

▶ 20. [*33*] 假定 MIX 的溢出开关和 X 寄存器已被连接到德尔玛大街和伯克利大道的交汇处的交通信号灯上，如下所示：

rX(2:2) = 德尔玛交通灯 }
rX(3:3) = 伯克利交通灯 } 0 关闭，1 绿色，2 黄色，3 红色

rX(4:4) = 德尔玛人行灯 }
rX(5:5) = 伯克利人行灯 } 0 关闭，1"WALK"(通过)，2"DON'T WALK"(不得通过)

希望穿过大街沿着伯克利大道行进的车辆和行人必须触动使 MIX 的溢出开关接通的一个开关。如果这个条件不出现，德尔玛的灯应保持为绿色。

循环时间如下：

德尔玛交通灯为绿色大于等于 30 s，黄色 8 s。

伯克利交通灯为绿色 20 s，黄色 5 s。

当一个方向的交通灯为绿色或黄色时，其它方向则为红色；当交通灯为绿色时，相应的 WALK 灯接通，此外在绿灯转为黄灯之前 DON'T WALK 灯闪亮 12 s，如下：

DON'T WALK $\frac{1}{2}$ s }
关闭 $\frac{1}{2}$ s } 重复 8 次

DON'T WALK　　4 s(而且在黄灯和红灯的整个周期内保持点亮)

如果当伯克利交通灯为绿色时溢出开关被接通,车辆或行人将在该周期里通过。但如果它是在交通灯为黄色或红色时被触动,则必须等候到在德尔玛上的交通已经通过之后的下一次循环。

假定一个 MIX 的时间单位是 10 μs。试编写一个完整的 MIX 程序,它按照由溢出开关给出的输入,通过对 rX 进行操作来控制这些灯。所述的时间应该精确地加以遵守,除非不可能这样做。注:rX 的设置精确地在一条 LDX 或 INCX 指令完成时改变。

21. [*28*] 一个 n 阶幻方就是在一个正方数组中对于从 1 到 n^2 的数的这样一种安排,即每行、每列以及两个主对角线之和都是 $n(n^2+1)/2$。图 16 示出了一个 7 阶的幻方。生成幻方的规则容易看出:在中间的方格的正下方以 1 开始,然后沿对角线往右下——当跑出这个边时想像整个平面都铺上方格——继续进行直到遇到被填充的方格为止;然后从新近填充的方格下移两格再继续。当 n 是奇数时,这个方法总有效。

22	47	16	41	10	35	04
05	23	48	17	42	11	29
30	06	24	49	18	36	12
13	31	07	25	43	19	37
38	14	32	01	26	44	20
21	39	08	33	02	27	45
46	15	40	09	34	03	28

图 16　幻方

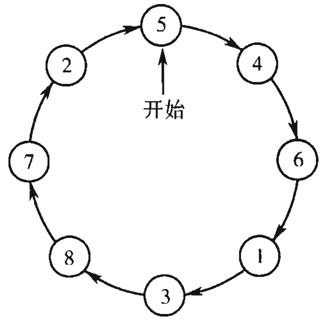


图 17　Josephus 问题 $n=8, m=4$

使用类似于习题 10 的内存分配方式,试编写一个完整的以上述方法生成 23×23 幻方的 MIX 程序,并把结果打印出来。[这个算法是由 Ibn al-Haytham 给出的,他于约 865 年生于巴士拉(Basra),而于约 1040 年死于开罗。关于其它许多有趣幻方的构造,其中有许多都是很好的程序设计题,请参见 W. W. Rouse Ball, *Mathematical Recreations and Essays*, H. S. M. Coxeter 修订(New York: Macmillan, 1939),第 7 章。]

22. [*31*] (Josephus 问题) 有 n 个男人被安排在一个圆圈上。从一个特定的位置开始,我们沿着这圆圈计数并且无情地处决每第 m 个人;当人死之后,这个圆圈就收缩。例如,当 $n=8$ 和 $m=4$时,如图 17 所示,处决的顺序是 5 4 6 1 3 8 7 2。头一个被处决的是第 5 号,第二个被处决的是第 4 号,等等。试写出当 $n=24, m=11$ 时打印出处决顺序的一个完整的 MIX 程序。当 n 和 m 都很大时,试设计出一个巧妙的高速算法(它可能救你的命)。参考文献:W. Ahrens, *Mathematische Unterhaltungen und Spiele* **2** (Leipzig: Teubner, 1918),第 15 章。

23. [37] 这道题旨在提供计算机的许多应用中的一些经验,在这些应用中,输出不是以通常的表格形式,而是以图形来加以显示。在现在的情况下,目标是"画出"一个纵横字谜的图形。

给你一个 0 和 1 为元素的矩阵作为输入。0 表示一个白的方格;1 表示一个黑的方格。输出应是一个字谜的图形,带有对纵横向字编了号的适当方格。

例如,给定矩阵为

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

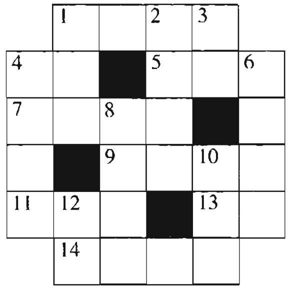


图 18　对应于习题 23 中矩阵的图形

它所对应的字谜图将如图 18 所示。一个方格,如果它是一个白方格而且或者(a)这个方格的下方是一个白方格且在它的紧上头没有白方格,或者(b)它右边的方格是白的而且紧靠它左边没有白方格,则它就被编号。如果黑方格出现在边上,则它们应从图中删除。这在图 18 中示出,其中在四个角处的黑方格已被删除。实现这一点的一个简单方法是在给定输入矩阵的顶上,下部以及边上人为地插入 - 1 的行和列,然后对于紧挨着 - 1 的每一个 + 1,把它改成 - 1,直到 - 1 旁边再无 + 1 存在为止。

下列方法将被用于在行式打印机上打印最后的图形。字谜的每一个方格对应于输出页的 5 列和 3 行。其中的 15 个位置如下填入:

未编号的	␣␣␣␣+	编号 nn 的	nn␣␣+	黑方格	+++++
白方格	␣␣␣␣+	白方格	␣␣␣␣+		+++++
	+++++		+++++		+++++

“ - 1”方格,依赖于是否有 - 1 在它右边或下边,而为

```
␣␣␣␣+   ␣␣␣␣+   ␣␣␣␣␣   ␣␣␣␣␣   ␣␣␣␣␣
␣␣␣␣+   ␣␣␣␣+   ␣␣␣␣␣   ␣␣␣␣␣   ␣␣␣␣␣
+++++   ␣␣␣␣+   +++++   ␣␣␣␣+   ␣␣␣␣␣
```

图 18 中所示的图形将被打印成如图 19 所示。

行式打印机的宽度——120 个字符——足以印出纵横字谜的 23 列。作为输入提供的数据将是以 0 和 1 为元素的 23 × 23 的矩阵,每一行被穿孔到一张输入卡片的 1 ~ 23 列中。例如,对应于上述矩阵的卡片被穿孔成“10000111111111111111111”。这个图形不必是对称的,而且它可能有以离奇的方式同外部连接的一长串黑方格。

```
     +++++++++++++++++++++
     +01  +    +02  +03  +
     +    +    +    +    +
+++++++++++++++++++++++++++++++
+04  +    ++++++05  +    +06  +
+    +    ++++++    +    +    +
+++++++++++++++++++++++++++++++
+07  +    +08  +    ++++++    +
+    +    +    +    ++++++    +
+++++++++++++++++++++++++++++++
+    ++++++09  +    +10  +    +
+    ++++++    +    +    +    +
+++++++++++++++++++++++++++++++
+11  +12  +    ++++++13  +    +
+    +    +    ++++++    +    +
+++++++++++++++++++++++++++++++
     +14  +    +    +    +
     +    +    +    +    +
     +++++++++++++++++++++
```

图 19　在一个行式打印机上图 18 的表示

1.3.3　对排列的应用

在这一小节里,我们将给出 MIX 程序更多的例子,同时介绍排列的某些重要性质。这些研究还将引出一般计算机程序设计的某些有趣方面。

早在 1.2.5 小节里,我们就已经讨论过排列。我们把排列 $c\ d\ f\ b\ e\ a$ 看成在一条直线上对于六个对象 a, b, c, d, e, f 的一种安排。还可以有另一种观点:把一个排列想像为对对象的一个重新安排或更名。对于这个解释,习惯上使用两行的记号。例如,

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \tag{1}$$

表示“a 变成 c，b 变成 d，c 变成 f，d 变成 b，e 变成 e，f 变成 a”。如果当做一个重新安排，这意味着对象 c 现在移动到以前由 a 所占据的位置；如果当做更名，它意味着对象 a 被更名为 c。两行记号不会因为列的顺序的改变而受影响；例如，排列(1)也可写成

$$\begin{pmatrix} c & d & f & b & a & e \\ f & b & a & d & c & e \end{pmatrix}$$

以及其它的 718 种方式。

同这种解释相联系，通常使用一种循环记号。排列(1)可以写成

$$(a\ c\ f)(b\ d) \tag{2}$$

这再一次意味着“a 变成 c，c 变成 f，f 变成 a，b 变成 d，d 变成 b”。循环$(x_1\ x_2\ \cdots\ x_n)$意味着“x_1 变成 x_2，…，x_{n-1}变成 x_n，x_n 变成 x_1”。由于在这个排列中 e 是固定的，因此在循环记号下它不出现；即，像“(e)”这样的单循环习惯上不写出来。如果一个排列使所有的元素都不动，以至仅有单一元素的循环出现，则它叫做*恒等排列*，而且就以“( )”来表示它。

循环的记号并不是惟一的。例如

$$(b\ d)(a\ c\ f),\quad (c\ f\ a)(b\ d),\quad (d\ b)(f\ a\ c) \tag{3}$$

等，都等价于(2)。然而“$(a\ f\ c)(b\ d)$”则不是同一个循环，因为它说的是 a 变成 f。

容易看出，为什么采用循环记号总是可能的。从任何元素 x_1 开始，比如说，排列把 x_1 变成 x_2，把 x_2 变成 x_3 等等，直到最后(因为仅仅有有限个元素)，我们得到已经在 $x_1,\cdots,x_n$ 当中出现过的某个元素 x_{n+1}。现在 x_{n+1}必定等于 x_1。因为如果它，比方说，等于 x_3，我们已经知道，x_2 变成 x_3。但根据假定 $x_n \neq x_2$ 变成 x_{n+1}。所以 $x_{n+1}=x_1$，因此我们有一个循环$(x_1\ x_2\cdots\ x_n)$，$n\geq 1$ 作为我们的排列的一部分；如果这还没有把整个排列包括在内，我们可以找出另一个元素 y_1，而且以同样的方式求得另一个循环$(y_1\ y_2\cdots\ y_m)$。这些 y 中没有一个能等于任何一个 x，因为 $x_i=y_j$ 意味着 $x_{i+1}=y_{j+1}$，等等，因而我们将最终地求出对于某个 k，$x_k=y_1$。这就同我们对 y_i 的选择相矛盾。以这种方式，最终将把所有循环都求出来。

每当以不同的顺序来放置 n 个对象的某个集合时，就会提出这些概念在程序设计中的应用。如果我们要重新安排这些对象而不把它们移动到别处去，就必须真正地采用循环结构。例如，为了重新安排(1)，即设置

$$(a,b,c,d,e,f) \leftarrow (c,d,f,b,e,a)$$

我们将真正地采用循环结构(2)并且逐次地置

$$t \leftarrow a, a \leftarrow c, c \leftarrow f, f \leftarrow t; t \leftarrow b, b \leftarrow d, d \leftarrow t。$$

任何这样的变换都是在不相交的循环中进行的，认识到这一点往往是很有用的。

排列的乘积　通过理解乘法意味着在一个排列之后再次应用另一个排列，我们可以把两个排列相乘起来。例如，如果在排列(1)之后，跟着做排列

$$\begin{pmatrix} a\,b\,c\,d\,e\,f \\ b\,d\,c\,a\,f\,e \end{pmatrix}$$

我们就有 a 变成c,然后 c 又变成c;b 变成d,它又变成 a,等等:

$$\begin{pmatrix} a\,b\,c\,d\,e\,f \\ c\,d\,f\,b\,e\,a \end{pmatrix} \times \begin{pmatrix} a\,b\,c\,d\,e\,f \\ b\,d\,c\,a\,f\,e \end{pmatrix} =$$

$$\begin{pmatrix} a\,b\,c\,d\,e\,f \\ c\,d\,f\,b\,e\,a \end{pmatrix} \times \begin{pmatrix} c\,d\,f\,b\,e\,a \\ c\,a\,e\,d\,f\,b \end{pmatrix} =$$

$$\begin{pmatrix} a\,b\,c\,d\,e\,f \\ c\,a\,e\,d\,f\,b \end{pmatrix} \tag{4}$$

很显然,排列的乘法不可以交换;换言之,当 π_1 和 π_2 是排列时,$\pi_1 \times \pi_2$ 未必等于 $\pi_2 \times \pi_1$。读者可以验证,在(4)中,如果把两个因子变换,则乘积给出不同的结果(参见习题 3)。

有些人不是以(4)中所示的稍微更自然的从左到右的顺序来进行排列的乘法,而是从右到左地进行。事实上,在这方面,数学家分成两派;应用变换 T_1,然后 T_2,结果应当记成 T_1T_2 呢,还是记成 T_2T_1? 这里我们使用 T_1T_2。

利用循环记号,等式(4)将写成如下:

$$(a\,c\,f)(b\,d)(a\,b\,d)(e\,f) = (a\,c\,e\,f\,b) \tag{5}$$

注意乘法的符号"×"习惯上被省略;这同循环的记号不冲突,因为容易看出排列$(a\,c\,f)(b\,d)$实际上就是排列$(a\,c\,f)$和$(b\,d)$的乘积。

排列的乘法可以直接地借助于循环记号来进行。例如,为了计算若干个排列

$$(a\,c\,f\,g)\,(b\,c\,d)\,(a\,e\,d)\,(f\,a\,d\,e)\,(b\,g\,f\,a\,e) \tag{6}$$

的乘积,我们发现(自左至右地进行),"a 变成c,然后 c 变成d,然后 d 变成a,然后 a 变成 d,然后 d 不变";所以在(6)之下其实际结果是 a 变成d;因此我们写下"$(a\,d$"作为部分答案。现在我们考虑对 d 的效果;"d 变成 b 变成 g";我们有"$(a\,d\,g$"的部分结果。考虑 g,我们发现现在"g 变成 a,变成 e,变成 f,变成 a",因此头一个循环关闭:"$(a\,d\,g)$"。现在我们挑还未曾出现的一个新元素,比如说 c;我们发现,c 变成 e,读者可以验证最终对于(6)得到的答案是"$(a\,d\,g)(c\,e\,b)$"。

现在让我们尝试用计算机来进行这一过程。以下的算法以能够实现机器计算的方式,表述在上一段中所描述的方法。

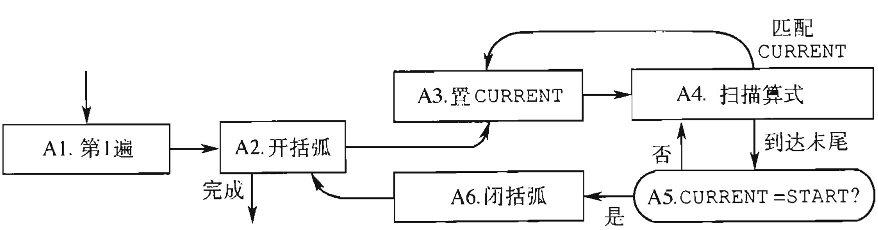


图 20 进行排列乘法的算法 A

算法 A(*以循环形式进行排列乘法*)　这一算法采用循环的乘积，如(6)那样，并以不相交的循环的乘积形式计算得到的排列。为简便起见，这里不描述单个元素循环的删除；那不过是本算法的一个相当简单的扩充而已。随着这一算法的实施，我们逐次地“标记”输入式的元素；即标记已被处理过的输入式的那些符号。

A1. [第 1 遍] 对所有左圆括弧加标记，并对每一个右圆括弧，用与之配对的左圆括弧后面的输入符号的加标记副本替换。(请参见表 1 中的例子。)

A2. [开括弧] 从左到右地查找，找出输入中头一个未曾标记的元素。(如果所有元素都已加了标记，则算法结束。)置 START 等于这个元素，输出一个左圆括弧，输出此元素并标记它。

A3. [置 CURRENT] 置 CURRENT 等于算式的下一个元素。

A4. [扫描算式] 向右进行直到或者达到算式的末尾，或者找到一个等于 CURRENT 的元素。在后一种情况下，对它加标记并返回步骤 A3。

A5. [CURRENT = START?] 如果 CURRENT ≠ START，输出 CURRENT 并返回到步骤 A4，再次由算式左边开始(从而继续扩展输出中的一个循环)。

A6. [闭括弧] (在输出中已经求出一个完整的循环。)输出一个右圆括弧，并返回步骤 A2。▌

例如，考虑式(6)；表 1 说明了在排列乘法处理过程中相继的各阶段。表的头一行

表 1　算法 A 应用于(6)

经本步骤后	START	CURRENT	(a c f g) (b c d) (a e d) (f a d e) (b g f a e)	输出
A1			(a c f g a (b c d b (a e d a (f a d e f (b g f a e b	
A2	a		(a⌶ c f g a (b c d b (a e d a (f a d e f (b g f a e b	(a
A3	a	c	(a c⌶ f g a (b c d b (a e d a (f a d e f (b g f a e b	
A4	a	c	(a c f g a (b c⌶ d b (a e d a (f a d e f (b g f a e b	
A4	a	d	(a c f g a (b c d b (a e d⌶ a (f a d e f (b g f a e b	
A4	a	a	(a c f g a (b c d b (a e d a (f a⌶ d e f (b g f a e b	
A5	a	d	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b⌶	d
A5	a	g	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b⌶	g
A5	a	a	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b⌶	
A6	a	a	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b⌶	)
A2	c	a	(a c⌶ f g a (b c d b (a e d a (f a d e f (b g f a e b	(c
A5	c	e	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b⌶	e
A5	c	b	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b⌶	b
A6	c	c	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b⌶	)
A6	f	f	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b⌶	(f)

符号 ⌶ 表示刚扫描的元素后面的光标；标记过的元素呈浅灰色

说明在右括弧已被相应循环的前导元素所代替之后的算式;这张表逐行说明随着越来越多的元素被标记,过程取得的进展。光标指出算式中当前感兴趣之点。输出是"$(adg)(ceb)(f)$"。注意在输出中将出现单元素的循环。

MIX **程序** 为了用 MIX 实现这个算法,"加标记"可通过使用一个字的符号来表示。假设我们的输入是以下列格式穿孔在卡片上的:一张 80 列的卡片被分成 16 个五字符的字段,每个字段或者是(a)"⊔⊔⊔⊔(",表示开始一个循环的左圆括弧;(b)")⊔⊔⊔⊔",表示结束一个循环的右圆括弧;(c)"⊔⊔⊔⊔⊔",全空白,它可以插在任何地方来填空;或者是(d)其它,表示待排列的一个元素。通过在 76~80 列中有"⊔⊔⊔⊔="来识别输入的最后一张卡片。例如,(6)可以穿孔到两张卡片上,如下:

```
 (  A  C  F  G  )    (  B  C  D  )    (  A  E  D  )
 (  F  A  D  E  )    (  B  G  F  A  E  )                          =
```

程序的输出将由输入的完全照字面的副本,紧跟着是格式上完全相同的答案所组成。

程序 A(*以循环形式进行排列乘法*) 本程序实现算法 A,而且它包括输入、输出部分以及删去单个元素的循环的部分,但输入并不捕获错误。

```
MAXWDS  EQU    1200                 输入的最大长度
PERM    ORIG   *+MAXWDS             输入的排列
ANS     ORIG   *+MAXWDS             答案位置
OUTBUF  ORIG   *+24                 打印位置
CARDS   EQU    16                   卡片输入机设备号
PRINTER EQU    18                   打印机设备号
BEGIN   IN     PERM(CARDS)          读头一张卡片
        ENT2   0
        LDA    EQUALS
1H      JBUS   *(CARDS)             等候循环完成
        CMPA   PERM+15,2
        JE     *+2                  是最后一张卡片吗?
        IN     PERM+16,2(CARDS)     否,读另一张
        ENT1   OUTBUF
        JBUS   *(PRINTER)           打印输入卡片的一个副本
        MOVE   PERM,2(16)
        OUT    OUTBUF(PRINTER)
        JE     1F
        INC2   16
        CMP2   =MAXWDS-16=
        JLE    1B                   重复直到输入完成
        HLT    666                  输入太多!
```

```
1H      INC2   15                1   这时,输入的 rI2 个字是
        ST2    SIZE              1     在 PERM,PERM+1,…中
        ENT3   0                 1   A1. 第 1 遍
2H      LDAN   PERM,3            A   取输入的下一个字符
        CMPA   LPREN(1:5)        A   是"("吗?
        JNE    1F                A
        STA    PERM,3            B   若是,对它加标记
        INC3   1                 B   置下一个非空白的符号于 rX 中
        LDXN   PERM,3            B
        JXZ    *-2               B
1H      CMPA   RPREN(1:5)        C
        JNE    *+2               C
        STX    PERM,3            D   以加了标记的 rX 替代")"
        INC3   1                 C
        CMP3   SIZE              C   所有元素都已处理了吗?
        JL     2B                C
        LDA    LPREN             1   为主程序做准备
        ENT1   ANS               1   rI1 = 存后面答案的位置
OPEN    ENT3   0                 E   A2. 开括弧
1H      LDXN   PERM,3            F   寻找未加标记的元素
        JXN    GO                F
        INC3   1                 G
        CMP3   SIZE              G
        JL     1B                G
*                                    全部加标记了,现在该输出了
DONE    CMP1   =ANS=
        JNE    *+2                   答案是恒等排列吗?
        MOVE   LPREN(2)              若是变成"()"
        MOVE   =0=                   在答案之后放置 23 个空白字
        MOVE   -1,1(22)
        ENT3   0
        OUT    ANS,3(PRINTER)
        INC3   24
        LDX    ANS,3                 按需要的行数打印
        JXNZ   *-3
        HLT
*
LPREN   ALF        (                 程序中所用常数
```

```
61 RPREN    ALF     )
62 EQUALS   ALF           =
63 *
64 GO       MOVE    LPREN            H   打开输出中的一个循环
65          MOVE    PERM,3           H
66          STX     START            H
67 SUCC     STX     PERM,3           J   标记一个元素
68          INC3    1                J   向右移一步
69          LDXN    PERM,3(1:5)      J   A3. 置 CURRENT(即 rX)
70          JXN     1F               J   跳过空格
71          JMP     *-3              0
72 5H       STX     0,1              Q   输出 CURRENT
73          INC1    1                Q
74          ENT3    0                Q   再次扫描算式
75 4H       CMPX    PERM,3(1:5)      K   A4. 扫描算式
76          JE      SUCC             K   元素 = CURRENT?
77 1H       INC3    1                L   右移
78          CMP3    SIZE             L   算式结束了吗?
79          JL      4B               L
80          CMPX    START(1:5)       P   A5 .CURRENT = START?
81          JNE     5B               P
82 CLOSE    MOVE    RPREN            R   A6. 关闭
83          CMPA    -3,1             R   注:rA = "("
84          JNE     OPEN             R
85          INC1    -3               S   删去单个元素的循环
86          JMP     OPEN             S
87          END     BEGIN                ▌
```

比起上一小节的程序来,这个大约有 75 行指令的程序要长得多了,而且它比起在本书中我们将遇到的大多数程序确实要更长些。然而它的长度并不可怕,因为它分成相当独立的若干小的部分。07 ~ 22 行读进输入卡片并且打印出每张卡片的副本;23 ~ 38 行实行算法的步骤 A1,即输入的预处理;39 ~ 46 行和 64 ~ 86 行完成算法 A 的主要工作;而 48 ~ 57 行输出答案。读者将会感到,尽其所能地研究本书中尽量多的 MIX 程序,将是有教益的——通过阅读他人的计算机程序获得技巧,是极其重要的,但在许许多多的计算机课程中这样的训练却可悲地被忽视了,因此导致了计算机极其糟糕地低效率的使用。

计时 程序 A 中不涉及输入输出的部分,已经辅之以频率计数,如同我们对程序 1.3.2M 所做的那样;因此,行 30 被假定执行 B 次。为方便起见,已经假定,除了在最右

端外，在输入中不出现空白字。在这一假设之下，行 71 根本不被执行，而且行 32 中的转移也从不发生。

通过简单的加法，得出执行这个程序的总时间是

$$(7 + 5A + 6B + 7C + 2D + E + 3F + 4G + 8H + 6J + 3K + 4L + 3P + 4Q + 6R + 2S)u \quad (7)$$

加上输入输出的时间。为了理解式(7)的意义，我们需要考察 15 个未知数 $A,B,C,D,E,F,G,H,J,K,L,P,Q,R,S$，而且我们必须把它们与输入的有关特征关联起来。现在我们将说明解决这种类型的问题的一般原理。

首先我们应用电路理论的"基尔霍夫第一定律"：一个指令被执行的次数必须等于我们转移到该指令的次数。这个看起来显然的规则往往以一种并不显然的方式把若干个量关联起来。分析程序 A 的流程，我们得到以下的方程：

从行	我们导出
26,38	$A = 1 + (C - 1)$
33,28	$C = B + (A - B)$
41,84,86	$E = 1 + R$
42,46	$F = E + (G - 1)$
64,43	$H = F - G$
67,70,76	$J = H + (K - (L - J))$
75,79	$K = Q + (L - P)$
82,72	$R = P - Q$

由基尔霍夫定律给出的这些方程并非全都是独立的。例如，在现在的情况下，我们看到，第一个和第二个方程明显地是等价的。其次，最后一个方程可从其它方程导出。因为第三、第四和第五个方程意味着 $H = R$；因此，第六个方程给出 $K = L - R$。无论如何，我们已经消去了 15 个未知数中的 6 个：

$$A = C,\ E = R + 1,\ F = R + G,\ H = R,\ K = L - R,\ Q = P - R \quad (8)$$

基尔霍夫第一定律是一个有效的工具，在 2.3.4.1 小节中我们将更仔细地分析它。

下一步是试图以数据的重要特征来匹配这些变量。我们由行 24,25,30 和 36 发现

$$B + C = \text{输入字的个数} = 16X - 1 \quad (9)$$

其中 X 是输入的卡片数。由行 28，

$$B = \text{输入中“(”的个数} = \text{输入中循环的个数} \quad (10)$$

类似地，从行 34，

$$D = \text{输入中“)”的个数} = \text{输入中循环的个数} \quad (11)$$

现在(10)和(11)给出了由基尔霍夫定律不能导出的一个事实：

$$B = D \quad (12)$$

由行 64，

$$H = \text{输出中循环(包括单个元素的循环)的个数} \quad (13)$$

行 82 指出 R 等于这相同的量；在此情况下，$H = R$ 这一事实是可从基尔霍夫定律导出的，因为在(8)中它已经出现了。

每一个非空白的字最终都会被标记。利用这一事实以及行 29,35 和 67,我们求得

$$J = Y - 2B \tag{14}$$

其中 Y 是出现于输入排列中的非空白字的个数。出现在输入排列中的每个不同元素写到输出中恰一次,或者在行 63 处,或者在行 72 处。由这一事实,我们得出(参见等式(8))

$$P = H + Q = \text{输入中不同元素的个数} \tag{15}$$

由对行 80 的短暂思考,也可使这一点成为显然的。最后,由行 85 我们看出

$$S = \text{输出中单个元素循环的个数} \tag{16}$$

显然,我们现在已予解释的量 B,C,H,J,P 和 S 实际上都是独立的参数,这些参数是预期要参与程序 A 的计时的。

迄今为止我们所得到的结果仅剩下未知的 G 和 L 有待分析。对于这些量,我们必须多使用一点技巧。由行 41 和 74 开始的对输入的扫描总是在行 47(最后一次)或者在行 80 结束。在这 $P+1$ 个循环的每一个当中,指令"INC3 1"被执行了 $B+C$ 次;这仅在行 44,68 和 77 处发生,所以我们得到连接未知的 G 和 L 的非平凡关系

$$G + J + L = (B + C)(P + 1) \tag{17}$$

正巧,运行时间(7)是 $G+L$ 的一个函数(它涉及$\cdots + 3F + 4G + \cdots + 3K + 4L + \cdots = \cdots + 7G + \cdots + 7L + \cdots$),因此我们不必试图再对单个的数量 G 和 L 作任何进一步的分析了。

综合以上所有这些结果,我们求得,除了输入输出之外,总共的时间计为

$$(112NX + 304X - 2M - Y + 11U + 2V - 11)u \tag{18}$$

在这个公式中,关于数据特征方面所使用的新名称如下:

$$\begin{aligned} X &= \text{输入卡片的张数} \\ Y &= \text{输入中非空白字段的个数(不包括最后的“=”)} \\ M &= \text{输入中循环的个数} \\ N &= \text{输入中不同元素名称的个数} \\ U &= \text{输出中循环(包括单个元素的循环)的个数} \\ V &= \text{输出中单个元素循环的个数} \end{aligned} \tag{19}$$

我们发现,按照这种方式分析程序 A 这样一类程序,在许多方面类似于猜解一个有趣的谜。

以下我们将说明,如果输出排列是随机的,则平均说来,量 U 和 V 分别是 H_N 和 1。

另一个解法 算法 A 把排列乘在一起,很像人们通常做的同样事情。我们常常发现,有待由计算机解决的问题,非常类似于多年来人类所面对的那些问题。因此,已经由像我们这样的世人发展起来加以应用的历史悠久的求解方法,对于计算机算法程序也是适当的。

然而,就如通常那样,我们也遇到对于计算机说来更优越的新方法,尽管它们十分不适合于人类使用。其核心的原因在于,计算机的"思维"方式是不同的;它有不同的记忆方法来记忆事实。这个差异的一个实例可以从我们的排列乘法问题看出:使用以下的算法,计算机可以在对算式的一次扫视中就把乘法做了,并且当它的循环相乘时计算

机就记住排列整个的当前状态。面向人类的算法 A 扫描算式许多次，对于输出的每一个元素都要扫描一次，但是新算法在一次扫描中就处理掉每件事，这是凡人（*Homo sapiens*）所不可能可靠地做到的一招绝技。

用于进行排列乘法的这个面向计算机的方法是怎样的呢？表 2 示出其基本思想。在这个表的循环形式中，每个字符以下的列指出由直到右边为止的部分循环所表示的排列是什么；例如，片断公式“$\cdots d\ e)\ (b\ g\ f\ a\ e)$”表示排列

$$\begin{pmatrix} a & b & c & d & e & f & g \\ e & g & c & b & ? & a & f \end{pmatrix}$$

它出现在这个表的最右边的 d 之下。

对表 2 的考察显示，如果我们从右边的恒等排列开始并且从右至左倒回来进行，就可系统地把这个表建立起来。在字母 x 之下的列仅在行 x 处不同于它右边的列（它记录以前的状态）；在行 x 处的新值是在上一次的改变中消失了的值。更精确地说，我们有下列的算法。

算法 B（循环形式的排列乘法）　这个算法实质上实现和算法 A 一样的结果。假定被排列的元素的名字是 $x_1, x_2, \cdots, x_n$。我们使用辅助表 $T[1], T[2], \cdots, T[n]$；在本算法结束时，当且仅当 $T[i]=j$ 时，在输入排列之下 x_i 变成 x_j。

表 2　在一次扫描中的排列的乘法

	(	a	c	f	g	)	(	b	c	d	)	(	a	e	d	)	(	f	a	d	e	)	(	b	g	f	a	e	)
a →	d	d	a	a	a	a	a	a	a	a	a	a	a	d	d	d	d	d	d	e	e	e	e	e	e	e	e	a	a
b →	c	c	c	c	c	c	c	c	g	g	g	g	g	g	g	g	g	g	g	g	g	g	g	g	b	b	b	b	b
c →	e	e	e	d	d	d	d	d	d	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
d →	g	g	g	g	g	g	g	)	)	)	d	d	)	)	)	b	b	b	b	b	d	d	d	d	d	d	d	d	d
e →	b	b	b	b	b	b	b	b	b	b	b	b	b	b	a	a	a	)	)	)	)	b	b	)	)	)	)	)	e
f →	f	f	f	f	e	e	e	e	e	e	e	e	e	e	e	e	e	e	a	a	a	a	a	a	a	a	f	f	f
g →	a	)	)	)	)	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	g	g	g	g

B1.［初始化］对于 $1 \leqslant k \leqslant n$，置 $T[k] \leftarrow k$。而且，准备从右到左地扫描输入。

B2.［下个元素］考察输入的下一元素（从右至左）。如果输入已经穷尽，则算法结束。如果元素是“)”，置 $Z \leftarrow 0$ 并重复步骤 B2；如果它是“(”，转到 B4。否则对于某个 i，元素是 x_i，转到 B3。

B3.［改变 $T[i]$］交换 $Z \leftrightarrow T[i]$。如果这使 $T[i]=0$，置 $j \leftarrow i$。返回步骤 B2。

B4.［改变 $T[j]$］置 $T[j] \leftarrow Z$。（这时，j 是在表 2 的记号下示出“)”项的行，并且对应于同刚刚被扫描的左圆括弧相匹配的右圆括弧。）返回步骤 B2。▌

当然，在执行了这个算法之后，我们仍然以循环形式输出表 T 的内容。如同我们在下边将看到的那样，通过一个“标记”方法，这很容易进行。

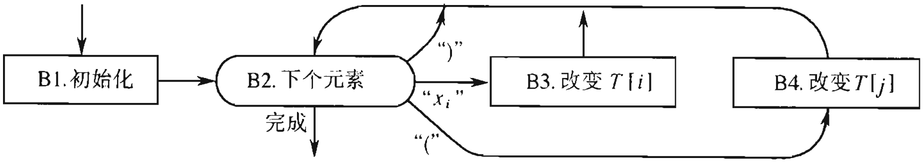


图 21　进行排列乘法的算法 B

现在我们编写一个基于这个新算法的 MIX 程序。我们希望使用和在程序 A 中相同的基本规则，而且输入和输出的格式与以前相同。但出现一点小问题，即如果事先不知道元素 $x_2, x_2, \cdots, x_n$ 是什么，那我们如何实现算法 B 呢？我们预先不知道 n，我们也不知道叫做 b 的元素是 x_1 呢，还是 x_2，等等。解决这个问题的一个简单方法是保留迄今为止已经遇到的元素的名称的表，而且每次去找当前的元素名(参见以下程序中的 35～44 行)。

程序 B(效果和程序 A 相同)　$rX \equiv Z$，$rI4 \equiv i$；$rI1 \equiv j$；$rI3 \equiv n$，即所见到的不同名称的数目。

```
01  MAXWDS   EQU    1200             输入的最大长度
02  X        ORIG   *+MAXWDS         名字表
03  T        ORIG   *+MAXWDS         辅助状态表
04  PERM     ORIG   *+MAXWDS         输入排列
05  ANS      EQU    PERM             答案位置
06  OUTBUF   ORIG   *+24             打印位置
07  CARDS    EQU    16         }
    ...                        }     和程序 A 的 05～22 行相同
24           HLT    666        }       这时输入的 rI2 个字是在
25  1H       INC2   15         1         PERM,PERM+1,…中,而且
26           ENT3   0          1         我们还未看到任何名字
27  RIGHT    ENTX   0          A     置 Z←0
28  SCAN     DEC2   1          B     B2. 下个元素
29           LDA    PERM,2     B
30           JAZ    CYCLE      B     跳过空白
31           CMPA   RPREN      C
32           JE     RIGHT      C     下个元素是")"吗?
33           CMPA   LPREN      D
34           JE     LEFT       D     是"("吗?
35           ENT4   1,3        E     为查找做准备
36           STA    X          E     存入表的开始处
37  2H       DEC4   1          F     查遍名字表
38           CMPA   X,4        F
```

```
39                JNE     2B          F   重复直到找到匹配为止
40                J4P     FOUND       G   这名称以前出现过没有?
41                INC3    1           H   否;增加表的大小
42                STA     X,3         H   插入新名称 x_n
43                ST3     T,3         H   置 T[n]←n
44                ENT4    0,3         H   i←n
45   FOUND        LDA     T,4         J   B3. 改变 T[i]
46                STX     T,4         J   存储 Z
47                SRC     5           J   置 Z
48                JANZ    SCAN        J
49                ENT1    0,4         K   如果 Z 为零,置 j←i
50                JMP     SCAN        K
51   LEFT         STX     T,1         L   B4. 改变 T[j]
52   CYCLE        J2P     SCAN        P   除非完成了,否则返回 B2
53   *
54   OUTPUT       ENT1    ANS         1   已扫描全部输入
55                J3Z     DONE        1   x 和 T 表包含答案
56   1H           LDAN    X,3         Q   现在我们构造循环记号
57                JAP     SKIP        Q   名字是否已标记?
58                CMP3    T,3         R   有一个单元素的循环吗?
59                JE      SKIP        R
60                MOVE    LPREN       S   开辟一个循环
61   2H           MOVE    X,3         T
62                STA     X,3         T   对名字加标记
63                LD3     T,3         T   寻找元素的后继者
64                LDAN    X,3         T
65                JAN     2B          T   是否已加了标记?
66                MOVE    RPREN       W   是,循环关闭
67   SKIP         DEC3    1           Z   移至下一个名字
68                J3P     1B          Z
69   *
70   DONE         CMP1    =ANS=       }
     ...                              }   和程序 A 的 48~62 行相同
84   EQUALS       ALF         =       }
85                END     BEGIN
```

从 T 表和名称表构造循环记号的 54 ~ 68 行,形成一个值得进行某些研究的堪称漂亮的小算法。参与这个程序计时的量 $A,B,\cdots,R,S,T,W,Z$ 当然不同于在程序 A 的分析

中具有相同名称的那些量。读者将会发现,分析这些计时将是饶有趣味的练习(参见习题 10)。

经验表明,程序 B 的执行时间的主要部分将花费在名称表的查找上——这就是在计时中的量 F。为查找和构造名称词典有好得多的算法可以利用;它们称做符号表算法,而且它们在计算机应用中有着巨大的重要性。第 6 章包含了关于高效符号表算法的全面讨论。

求逆 一个排列 π 的逆 π^- 是一个撤消 π 的结果的重新安排;如果在 π 之下 i 变成 j,则在 π^- 之下 j 变成 i。因此 $\pi\pi^-$ 等于恒等排列,乘积 $\pi^-\pi$ 也是这样。

人们经常以 π^{-1} 来表示逆,而不是用 π^-。但上标 1 是多余的(和 $x^1=x$ 的原因相同)。

每一个排列有一个逆。例如,

$$\begin{pmatrix} a\,b\,c\,d\,e\,f \\ c\,d\,f\,b\,e\,a \end{pmatrix} \quad \text{的逆是} \quad \begin{pmatrix} c\,d\,f\,b\,e\,a \\ a\,b\,c\,d\,e\,f \end{pmatrix} = \begin{pmatrix} a\,b\,c\,d\,e\,f \\ f\,d\,a\,b\,e\,c \end{pmatrix}$$

现在我们将考虑计算一个排列的逆的简单算法。

在本小节的剩余部分,我们假定正在处理数 $\{1,2,\cdots,n\}$ 的排列。若 $X[1]X[2]\cdots X[n]$ 是这样的一个排列,则有一个简单的方法来计算它的逆:对于 $1\leqslant k\leqslant n$,置 $Y[X[k]]\leftarrow k$。于是 $Y[1]Y[2]\cdots Y[n]$ 就是所求的逆。这个方法使用 $2n$ 个内存单元,即 X 用 n 个和 Y 用 n 个。

然而,就为好玩吧,假设 n 是非常大的,而且还假设,我们希望计算 $X[1]X[2]\cdots X[n]$ 的逆而不使用许多另外的内存空间。我们要来"就地"计算逆,使得在我们的算法完成之后,数组 $X[1]X[2]\cdots X[n]$ 将是原来的排列的逆。对于 $1\leqslant k\leqslant n$ 仅仅置 $X[X[k]]\leftarrow k$ 肯定不灵,但通过循环结构我们可以推导出下列简单的算法:

算法 I(*就地求逆*) 将 $\{1,2,\cdots,n\}$ 的一个排列 $X[1]X[2]\cdots X[n]$ 用它的逆代替。这个算法是由黄秉超(Bing-Chao Huang)给出的[*Inf. Proc. Letters* **12** (1981),237~238]。

I1. [初始化] 置 $m\leftarrow n, j\leftarrow -1$。

I2. [下个元素] 置 $i\leftarrow X[m]$。如果 $i<0$,转到步骤 I5(此元素已经处理过)。

I3. [对一个元素求逆] (这时 $j<0$ 和 $i=X[m]$。如果 m 不是它的循环的最大元素,则原来的排列有 $X[-j]=m$。)置 $X[m]\leftarrow j, j\leftarrow -m, m\leftarrow i, i\leftarrow X[m]$。

I4. [循环结束了吗?] 如果 $i>0$,转回 I3 (循环还未结束);否则置 $i\leftarrow j$。(在后一情况下,原来的排列有 $X[-j]=m$ 且 m 是在它的循环中之最大者。)

I5. [存储最后的值] 置 $X[m]\leftarrow -i$。(原来 $X[-i]$ 等于 m。)

I6. [对 m 循环] m 减 1。如果 $m>0$,转回 I2;否则算法结束。 ▌

关于此算法的一个例子参见表 3。这个算法是基于排列的逐个循环的求逆,通过使元素取负来标记被求逆的元素,之后再恢复正确的符号。

表 3　由算法 I 计算 6 2 1 5 4 3 的逆

经本步骤后	I2	I3	I3	I3	I5 *	I2	I3	I3	I5	I2	I5	I5	I3	I5	I5
$X[1]$	6	6	6	−3	−3	−3	−3	−3	−3	−3	−3	−3	−3	−3	3
$X[2]$	2	2	2	2	2	2	2	2	2	2	2	2	−4	2	2
$X[3]$	1	1	−6	−6	−6	−6	−6	−6	−6	−6	−6	6	6	6	6
$X[4]$	5	5	5	5	5	5	5	−5	−5	−5	5	5	5	5	5
$X[5]$	4	4	4	4	4	4	−1	−1	4	4	4	4	4	4	4
$X[6]$	3	−1	−1	−1	1	1	1	1	1	1	1	1	1	1	1
m	6	3	1	6	6	5	4	5	5	4	4	3	2	2	1
j	−1	−6	−3	−1	−1	−1	−5	−4	−4	−4	−4	−4	−2	−2	−2
i	3	1	6	−1	−1	4	5	−1	−4	−5	−5	−6	−4	−2	−3

从左到右读各列，在 * 处，循环(1 6 3)已经求出了逆

算法 I 类似算法 A 的一些部分，而且它非常类似程序 B 中的求循环的算法(54～68 行)，因此它在涉及重新安排的一些算法中是很典型的。当编写 MIX 的实现时，我们发现，在一个寄存器中保存 $-i$ 而不是 i 本身是最方便的。

程序 I(就地求逆)　rI1 ≡ m；rI2 ≡ $-i$；rI3 ≡ j；以及 n = N，即当这个程序被汇编作为一个更大程序的一部分时，要加以定义的一个符号。

```
INVERT  ENT1   N     1   I1. 初始化。 m←n
        ENT3   -1    1   j←-1
2H      LD2N   X,1   N   I2. 下个元素。 i←X[m]
        J2P    5F    N   如果 j<0 转 I5
3H      ST3    X,1   N   I3. 对一个元素求逆。 X[m]←j
        ENN3   0,1   N   j←-m
        ENN1   0,2   N   m←i
        LD2N   X,1   N   i←X[m]
4H      J2N    3B    N   I4. 循环结束了吗？ 如果 i>0 转 I3
        ENN2   0,3   C   否则置 i←j
5H      ST2    X,1   N   I5. 存最后的值。 X[m]←-i
6H      DEC1   1     N   I6. 对 m 循环
        J1P    2B    N   如果 m>0 转 I2  ▌
```

这个程序的计时，按前边所示的方式，很容易进行。每一个元素 $X[m]$ 首先在步骤 I3 中被置成为负的值。而后在步骤 I5 再置成为正的值。总共的时间成为$(14N + C + 2)u$，其中 N 是数组的大小，而 C 是循环的全部个数。下面分析在一个随机的排列中 C 的特性。

对于完成任何给定的任务，几乎总有一个以上的算法，因此我们预期可能还会有另外的求一个排列的逆的方法。下边这个巧妙的算法是 J. Boothroyd 给出的。

算法 J(就地求逆)　这个算法有和算法 I 一样的效果但使用不同的方法。

J1. [全部取负] 对于 $1 \leqslant k \leqslant n$,置 $X[k] \leftarrow -X[k]$。而且置 $m \leftarrow n$。

J2. [初始化 j] 置 $j \leftarrow m$。

J3. [求负的项] 置 $i \leftarrow X[j]$。如果 $i > 0$,置 $j \leftarrow i$ 并重复这一步骤。

J4. [求逆] 置 $X[j] \leftarrow X[-i]$,$X[-i] \leftarrow m$。

J5. [对 m 循环] m 减 1;如果 $m > 0$,转回 J2。否则算法结束。 ▌

表 4　通过算法 J 计算 6 2 1 5 4 3 的逆

经本步骤后:	J2	J3	J5	J3	J5	J3	J5	J3	J5	J3	J5	J3	J5
$X[1]$	−6	−6	−6	−6	−6	−6	−6	−6	3	3	3	3	3
$X[2]$	−2	−2	−2	−2	−2	−2	−2	−2	−2	−2	2	2	2
$X[3]$	−1	−1	6	6	6	6	6	6	6	6	6	6	6
$X[4]$	−5	−5	−5	−5	5	5	5	5	5	5	5	5	5
$X[5]$	−4	−4	−4	−4	−5	−5	4	4	4	4	4	4	4
$X[6]$	−3	−3	−1	−1	−1	−1	−1	−1	−6	−6	−6	−6	1
m	6	6	5	5	4	4	3	3	2	2	1	1	0
i		−3	−3	−4	−4	−5	−5	−1	−1	−2	−2	−6	−6
j	6	6	6	5	5	5	5	6	6	2	2	6	6

关于 Boothroyd 算法的一个例子,参见表 4。这个方法实际上仍是基于循环结构,但这次这个算法真的有效却不大明显。其正确性验证留给读者(参见习题 13)。

程序 J(类似于程序 I)　rI1 ≡ m; rI2 ≡ j; rI3 ≡ $-i$。

```
INVERT  ENN1  N               1   J1. 全部取负
        ST1   X+N+1,1(0:0)    N   置负的符号
        INC1  1               N
        J1N   *-2             N   还有吗?
        ENT1  N               1   m←n
2H      ENN3  0,1             N   J2. 初始化 j。i←m
        ENN2  0,3             A   j←i
        LD3N  X,2             A   J3. 求负的项
        J3N   *-2             A   i>0?
        LDA   X,3             N   J4. 求逆
        STA   X,2             N   X[j]←X[-i]
        ST1   X,3             N   X[-i]←m
        DEC1  1               N   J5. 对 m 循环
        J1P   2B              N   如果 m>0,转 J2  ▌
```

为了判定这个算法的运行有多快,我们需要知道量 A;这个量极其有趣和有教益,因而已被留作习题(参见习题 14)。

尽管算法 J 是极其巧妙的,但分析表明,算法 I 肯定更优越。事实上,算法 J 的平均运行时间基本上同 $n \ln n$ 成比例,而算法 I 的平均运行时间实际上同 n 成比例。或许

某一天有人将发现算法 J(或某个相关的修改)的某个用途;否则全然把它忘了是太可惜了!

一个不寻常的对应　我们已经指出,一个排列的循环记号并非惟一;六个元素的排列(1 6 3)(4 5)可以写成(5 4)(3 1 6),等等。对于循环记号考虑一个范式(canonical form)将是有用的;范式是惟一的。为获得范式,如下进行:

a)直接写出所有的单元素循环。

b)在每一个循环内,首先写出最小的元素。

c)以循环中头一个元素的递减顺序来对诸循环排序。

例如,以(3 1 6)(5 4)开始,我们将得到

$$(a):(3\ 1\ 6)\ (5\ 4)\ (2);\ (b):(1\ 6\ 3)\ (4\ 5)\ (2);\ (c):(4\ 5)\ (2)\ (1\ 6\ 3) \qquad (20)$$

这个范式的重要性质是:圆括弧可以去掉而且可以惟一地再重新将其构造出来。因此,只有一种方法往“4 5 2 1 6 3”当中插入圆括弧以得到循环范式。必须在每个自左到右的极小值的紧前边插上左圆括弧(即左圆括弧必须括在其前边没有更小的元素的紧前边)。

圆括弧的插入和删除给了我们在以循环形式表达的所有排列的集合与以线性形式表达的所有排列的集合之间不寻常的一一对应。例如,在范式之下的排列6 2 1 5 4 3是(4 5) (2) (1 6 3);删除圆括号得到4 5 2 1 6 3,在循环形式下它是(2 5 6 3) (1 4);删去括弧得到2 5 6 3 1 4,在循环形式下它是(3 6 4) (1 2 5);等等。

这个对应对于研究不同类型的排列有许多应用。例如,如果我们问“平均说来,n 个元素的排列共有多少个循环”,为了回答这个问题,我们考虑在范式之下表达的所有 $n!$个排列的集合,并去掉圆括弧。我们得到在某种顺序下的所有 $n!$个排列的集合。因此,我们原来的问题等价于“平均说来,n 个元素的一个排列有多少个自左到右的极小值”。在 1.2.10 小节中我们已经对这后一问题作了回答,这就是在算法 1.2.10M 的分析中的量($A+1$),对于它我们求得统计

$$\min 1,\quad \text{ave } H_n,\quad \max n,\quad \text{dev} \sqrt{H_n - H_n^{(2)}} \qquad (21)$$

(实际上,我们讨论了自右到左的极大值的平均数,但这显然和自左到右的极小值的个数相同。)而且,我们实际上证明了,n 个对象的一个排列以 $\left[{n \atop k}\right]\Big/ n!$ 的概率,有 k 个自左到右的极小值;因此 *n 个对象的一个排列以 $\left[{n \atop k}\right]\Big/ n!$ 的概率有 k 个循环*。

我们也可以问在自左到右的极小值之间的平均距离,它变成等价于一个循环的平均长度。由(21),在所有$n!$ 个排列中循环的总数是 $n!H_n$,因为它等于 $n!$ 乘以循环的平均数。如果我们随机地选出这些循环之一,它的平均长度是多少?

想像以循环记号写下的$\{1,2,\cdots,n\}$的所有 $n!$ 个排列。有多少个三元循环出现?为了回答这个问题,让我们考虑一个特定的三元循环($x\ y\ z$)出现多少次:显然它恰好出现在($n-3$)! 个排列中,因为这是剩下的 $n-3$ 个元素可以被排列的方式数。现在不同可能的三元循环($x\ y\ z$)的个数是 $n(n-1)(n-2)/3$,因为对于 x 有 n 个选择,对于 y

有$(n-1)$个选择，对于 z 有$(n-2)$个选择。而在这些 $n(n-1)(n-2)$种选择当中，每个不同的三元循环已出现在三个形式$(x\ y\ z)$ $(y\ z\ x)$ $(z\ x\ y)$中。因此在所有$n!$ 个排列中三元循环的总数是 $n(n-1)(n-2)/3$ 乘以$(n-3)!$，即是$n!/3$。类似地，对于$1\leq m\leq n$，m 元循环的总数是 $n!/m$。(这提供了关于循环的总数是$n!\ H_n$这一事实的另一个简单的证明。因此如同我们已经知道的那样，在一个随机排列之下循环的平均个数是 H_n。)习题 17 证明一个随机地选择的循环的平均长度是 n/H_n，如果我们认为 $n!\ H_n$ 元循环是有相同概率的话；但如果我们在一个随机的排列中随机地选择一个元素，则包含该元素的循环的平均长度比 n/H_n 稍微大些。

为了完成我们对算法 A 和 B 的分析，我们想要知道在一个随机排列中单个元素的循环的平均个数。这是一个有趣的问题。假如我们把 $n!$ 个排列写下来，首先列出那些没有单个元素循环，然后是那些包含有一个单个元素的循环等等；例如，如果 $n=4$，

无固定元素：2143　2341　2413　3142　3412　3421　4123　4312　4321
一个固定元素：1342　1423　3241　4213　2431　4132　2314　3124
两个固定元素：1243　1432　1324　4231　3214　2134
三个固定元素：
四个固定元素：1234

(单个元素的循环，是在一个排列中保持固定不变的那些元素，在这个表中已被特别地标出。)无固定元素的那些排列称为*更列*(derangement)，更列的个数是把 n 封信放进n 个信封且让它们全都放错的方式数。

设 P_{nk}是n 个对象恰有 k 个元素保持固定的排列个数。例如，

$$P_{40}=9,\quad P_{41}=8,\quad P_{42}=6,\quad P_{43}=0,\quad P_{44}=1$$

对上边表的考察揭示出这些数之间的关系：我们可以通过首先选择要加以固定的 k(这可以$\binom{n}{k}$种方式来完成)，然后对于不再保持固定的剩下的 $n-k$ 个元素，以所有$P_{(n-k)0}$种方式进行排列，而得到有 k 个固定元素的所有排列。因此

$$P_{nk}=\binom{n}{k}P_{(n-k)0} \tag{22}$$

我们还有“全体是其部分之和”这样一条规则：

$$n!=P_{nn}+P_{n(n-1)}+P_{n(n-2)}+P_{n(n-3)}+\cdots \tag{23}$$

把公式(22)和(23)组合在一起并且稍微改写一下所得结果，我们求得

$$n!=\frac{P_{00}}{0!}+n\frac{P_{10}}{1!}+n(n-1)\frac{P_{20}}{2!}+n(n-1)(n-2)\frac{P_{30}}{3!}+\cdots \tag{24}$$

这是一个对于所有正整数 n 必定为真的等式。这个等式在以前我们就已碰到过——它出现于 1.2.5 小节有关斯特林试图推广阶乘函数的讨论中——而且在 1.2.6 小节中我们求出过它的系数的一个简单推导(例 5)。我们得到

$$\frac{P_{m0}}{m!}=1-\frac{1}{1!}+\frac{1}{2!}-\cdots+(-1)^m\frac{1}{m!} \tag{25}$$

现在命 p_{nk}是n 个对象的排列恰有 k 个单元素循环的概率。由于 $p_{nk}=P_{nk}/n!$，从

(22)和(25)我们有

$$p_{nk}=\frac{1}{k!}\left(1-\frac{1}{1!}+\frac{1}{2!}-\cdots+(-1)^{n-k}\frac{1}{(n-k)!}\right) \tag{26}$$

生成函数 $G_n(z)=p_{n0}+p_{n1}z+p_{n2}z^2+\cdots$因此是

$$G_n(z)=1+\frac{1}{1!}(z-1)+\cdots+\frac{1}{n!}(z-1)^n=\sum_{0\leqslant j\leqslant n}\frac{1}{j!}(z-1)^j \tag{27}$$

由这个公式得出 $G'_n(z)=G_{n-1}(z)$,而且使用 1.2.10 小节的方法,我们得到关于单元素循环的下列统计

$$(\min 0,\quad \text{ave } 1,\quad \max n,\quad \text{dev } 1),\qquad 若\quad n\geqslant 2 \tag{28}$$

计算没有单元素循环的排列个数的一个稍微更直接的方法是由容斥原理得出的。该原理是解决许多枚举问题的一个重要方法。容斥的一般原理可以表述如下:给定 N 个元素和这些元素的 M 个子集 $S_1,S_2,\cdots,S_M$;我们的目标是要计算不在这些子集中的元素有多少。令 $|S|$ 表示在一个集合 S 中的元素的个数,则所求的不在集合 S_j 中的对象的个数是

$$N-\sum_{1\leqslant j\leqslant M}|S_j|+\sum_{1\leqslant j<k\leqslant M}|S_j\cap S_k|-\sum_{1\leqslant i<j<k\leqslant M}|S_i\cap S_j\cap S_k|+\cdots$$
$$+(-1)^M|S_1\cap\cdots\cap S_M| \tag{29}$$

(这样,我们首先从总数 N 中减去在 $S_1,\cdots,S_M$ 中的元素的数目;但这就把欲求的总数低估了。所以我们还须把同属于一对集合,即对于每对 S_j 和 S_k,属于 $S_j\cap S_k$ 的元素个数加回去;然而,这又产生高估了,所以又减去为三个集合共有的元素,等等)。证明这个公式的方法有好多种,请读者也来参与发现其一(参见习题 25)。

为了计算没有单元素循环的 n 个元素的排列的数目,我们考虑 $N=n!$ 个排列并令 S_j 是元素 j 形成一单元素循环的排列之集合。如果 $1\leqslant j_1<j_2<\cdots<j_k\leqslant n$,则在 $S_{j_1}\cap S_{j_2}\cap\cdots\cap S_{j_k}$ 中的元素的数目就是其中 $j_1,\cdots,j_k$ 是单元素循环的排列的数目,而这显然是 $(n-k)!$。因此公式(29)变成

$$n!-\binom{n}{1}(n-1)!+\binom{n}{2}(n-2)!-\binom{n}{3}(n-3)!+\cdots+(-1)^n\binom{n}{n}0!$$

这同(25)一致。

容斥原理是由 A. de Moivre 给出的[参见他所著的*Doctrine of Chances* (London:1718),61~63;第 3 版(1756,由 Chelsea 重印,1957),110~112],但它的意义在 I. Todhunter 于其 *Algebra* (第 2 版,1860),§762,以及 W. A. Whitworth 于其名著*Choice and Chance* (Cambridge:1867)中对之宣传并作进一步发展之前,并未被普遍地理解。

在 5.1 节中进一步剖析排列的组合性质。

习　题

1. [*02*] 考虑以 $2x \bmod 7$ 代替 x 的 $\{0,1,2,3,4,5,6\}$ 的变换。证明这个变换是一个排列,并以循环形式把它写出来。

2. [*10*] 正文中说明我们如何通过使用一系列的替换运算($x \leftarrow y$)以及一个辅助变量 t,设置$(a,b,c,d,e,f) \leftarrow (c,d,f,b,e,a)$。说明如何使用一系列的交换运算($x \leftrightarrow y$)而无辅助变量,来做这件事。

3. [*03*] 计算乘积$\begin{pmatrix} a\ b\ c\ d\ e\ f \\ b\ d\ c\ a\ f\ e \end{pmatrix} \times \begin{pmatrix} a\ b\ c\ d\ e\ f \\ c\ d\ f\ b\ e\ a \end{pmatrix}$,并以两行记号来表达答案(同(4)作比较)。

4. [*10*] 把$(a\ b\ d)\ (e\ f)\ (a\ c\ f)\ (b\ d)$表达为不相交循环的乘积。

▶ 5. [*M10*] 等式(3)示出以循环形式表达同一个排列的若干个等价的方法。如果去掉所有的单元素循环,写出该排列有多少种可能的不同方法。

6. [*M28*] 如果我们去掉所有空白字出现于极右边的假定,对于程序 A 的计时要做哪些改变?

7. [*10*] 如果程序 A 和输入(6)一起给出,(19)中的量 X, Y, M, N, U 和 V 是什么?不包括输入输出,程序 A 所要求的时间是多少?

▶ 8. [*23*] 把程序 B 的输入过程修改成从左至右地进行而不是从右至左,是否能行?

9. [*10*] 程序 A 和程序 B 接受相同的输入而且以实质上相同的形式给出答案。在两个程序下,输出完全相同吗?

▶ 10. [*M28*] 考察程序 B 的计时特征,即程序中所示的量 $A, B, \cdots, Z$。借助于在(19)中定义的量 X, Y, M, N, U, V 并借助 F,表达总的时间。如同在习题 7 中所计算的那样,比较对于输入(6),程序 B 总的运行时间和程序 A 总的运行时间。

11. [*15*] 如果以循环形式给出排列 π,试求出以循环形式写出 π^- 的简单规则。

12. [*M27*](转置一个矩形矩阵) 假设以类似于习题 1.3.2-10 的方式在内存中储存一个 $m \times n$ 矩阵(a_{ij}),$m \neq n$,使得 a_{ij}的值出现于单元 $L + n(i-1) + (j-1)$中,其中 L 是 a_{11}的位置。问题是找出转置这个矩阵的一个方法,得到一个 $n \times m$ 矩阵(b_{ij}),其中 $b_{ij} = a_{ji}$被存储于单元 $L + m(i-1) + (j-1)$中。因此此矩阵"在它自己上"被转置。(a)证明对于范围 $0 \leqslant x < N = mn - 1$ 中的所有 x,转置转换把出现于单元 $L + x$ 中的值移动到单元 $L + (mx \bmod N)$中。(b)试讨论通过计算机来进行这个转置的方法。

▶ 13. [*M24*] 证明算法 J 是正确的。

▶ 14. [*M34*] 在算法 J 的计时之下求量 A 的平均值。

15. [*M12*] 是否有这样的排列存在,它在不带圆括弧的范式和在线性形式之下表示完全相同的转置?

16. [*M15*] 以在线性记号下的排列 1 3 2 4 开始;把它转换成循环范式而后删去圆括弧;重复这一过程直到达到原来的排列为止。在这过程中,出现什么排列?

17. [*M24*] (a)正文证明了,在 n 个元素的所有排列中,共有 $n!\,H_n$ 个循环,如果把这些循环(包括单元素循环在内)写在 $n!\,H_n$ 张纸上,而且如果随机地选择这些纸张中的一张,由此被选择的循环的平均长度是多少?(b) 如果我们把 $n!$ 个排列写到 $n!$ 张纸上,而且如果随机地选一个数 k 和选一张纸,包含元素 k 的一个循环是一个 m 元循环的概率是多少?包含元素 k 的循环的平均长度是多少?

▶ 18. [*M27*] n 个对象的一个排列恰有长度为 m 的 k 个循环的概率 p_{nkm}是多少?相应的生成函数 $G_{nm}(z)$是什么?m 元循环的平均个数是什么?标准差是多少?(正文中仅仅考虑了 $m = 1$ 的情况。)

19. [*HM21*] 在等式(25)的记号下,证明对于所有 $n \geqslant 1$,更列的个数 P_{n0}恰等于 $n!/e$ 舍入到最接近的整数。

20.[*M20*] 假定所有单元素的循环都被明确地写出，问写出有 α_1 个单循环，α_2 个二元循环，…的一个排列的循环记号有多少种不同的方法？

21.[*M22*] n 个对象的一个排列恰有 α_1 个单循环，α_2 个二元循环等的概率 $P(n;\alpha_1,\alpha_2,\cdots)$ 是多少？

▶ 22.[*HM34*] (由 L. Sheep 和 S. P. Lloyd 提出的下列方法，给出了解决有关随机排列之循环结构问题的一个方便和强有力的方法。)不再把对象的个数 n 当成是固定的，排列是可变的，现在我们假定，按照某种概率分布，我们独立地选择出现在习题 20 和 21 中的量 $\alpha_1,\alpha_2,\alpha_3,\cdots$。设 w 是 0 和 1 之间的任何实数。

a)假设我们按照以下规则，即对于某个函数 $f(w,m,k)$，"$\alpha_m=k$ 的概率是 $f(w,m,k)$"来选择随机变量 $\alpha_1,\alpha_2,\alpha_3,\cdots$。试确定 $f(w,m,k)$ 的值使得以下两个条件成立：(i) 对于 $0<w<1$ 和 $m\geqslant 1$，$\sum_{k\geqslant 0}f(w,m,k)=1$。(ii) $\alpha_1+2\alpha_2+3\alpha_3+\cdots=n$ 和 $\alpha_1=k_1,\alpha_2=k_2,\alpha_3=k_3,\cdots$ 的概率等于 $(1-w)w^nP(n;k_1,k_2,k_3,\cdots)$，其中 $P(n;k_1,k_2,k_3,\cdots)$ 在习题 21 中定义。

b)循环结构为 $\alpha_1,\alpha_2,\alpha_3,\cdots$ 的一个排列显然恰好排列 $\alpha_1+2\alpha_2+3\alpha_3+\cdots$ 个对象。证明如果诸 α 是按照 a)中的概率分布随机地选定的，则 $\alpha_1+2\alpha_2+3\alpha_3+\cdots=n$ 的概率是 $(1-w)w^n$；$\alpha_1+2\alpha_2+3\alpha_3+\cdots$ 是无穷的概率为零。

c)令 $\phi(\alpha_1,\alpha_2,\cdots)$ 是无穷多个数 $\alpha_1,\alpha_2,\cdots$ 的任意函数，试证明如果诸 α 是按照 a)中的概率分布选择的，则 ϕ 的平均值是 $(1-w)\sum_{n\geqslant 0}w^n\phi_n$；这里 ϕ_n 表示对于 n 个对象的所有排列所取的 ϕ 的平均值，其中变量 α_j 表示一个排列的 j 元循环的个数。[例如，如果 $\phi(\alpha_1,\alpha_2,\cdots)=\alpha_1$，则 ϕ_n 的值是 n 个对象的一个随机排列中单元素循环的平均个数；在(28)中我们证明了，对于所有 n，$\phi_n=1$。]

d)使用这个方法求在 n 个对象的一个随机排列中偶数长度的循环的平均个数。

e)使用这一方法求解习题 18。

23.[*HM42*] (Golomb, Shepp, Lloyd) 如果 l_n 表示在 n 个对象的一个排列中最长循环的平均长度，试证明 $l_n\approx\lambda n+\frac{1}{2}\lambda$，其中 $\lambda\approx 0.62433$ 是一个常数。试证明，事实上 $\lim_{n\to\infty}(l_n-\lambda n-\frac{1}{2}\lambda)=0$。

24.[*M41*] 试求进入算法 J 的计时的量 A 的方差(参见习题 14)。

25.[*M22*] 证明等式(29)。

▶ 26.[*M24*] 推广容斥原理来得出恰在子集 $S_1,S_2,\cdots,S_M$ 中的 r 个子集当中的元素个数的公式。(正文仅考虑了 $r=0$ 的情况。)

27.[*M20*] 使用容斥原理计算在 $0\leqslant n<am_1m_2\cdots m_t$ 中不为 $m_1,m_2,\cdots,m_t$ 的任何一个所整除的整数 n 的个数。这里 $m_1,m_2,\cdots,m_t$ 和 a 都是正整数，且当 $j\neq k$ 时 $m_j\perp m_k$。

28.[*M21*] (I. Kaplansky) 如果把在习题 1.3.2-22 中定义的"Josephus 排列"表达成循环形式，当 $n=8$ 和 $m=4$ 时我们得到(1 5 3 6 8 2 4) (7)。证明在一般情况下这个排列是 $(n\ n-1\ \cdots\ 2\ 1)^{m-1}\times(n\ n-1\ \cdots\ \dot{2})^{m-1}\cdots(n\ n-1)^{m-1}$ 的乘积。

29.[*M25*] 证明当 $m=2$ 时 Josephus 排列的循环形式可以通过首先表达 $\{1,2,\cdots,2n\}$ 的"加倍"排列成循环形式(它把 j 变成 $(2j)\bmod(2n+1)$)，然后把左和右颠倒并删除所有大于 n 的数来得到。例如，当 $n=11$ 时，加倍排列是(1 2 4 8 16 9 18 13 3 6 12) (5 10 20 17 11 22 21 19 15 7 14)，因而 Josephus 排列是(7 11 10 5) (6 3 9 8 4 2 1)。

30.[*M24*] 利用习题 29 证明，当 $m=2$ 时的 Josephus 排列的固定元素恰是对于所有正整数

d，使得$(2^d-1)(2n+1)/(2^{d+1}-1)$为整数的那些值

31. [*HM33*] 推广习题29和30，证明对于一般的 m 和 n，被处决的第 k 个人，是位于位置 x 的人。其中对 x 的计算如下：置 $x \leftarrow km$，然后重复地置 $x \leftarrow \lfloor (m(x-n)-1)/(m-1) \rfloor$ 直到 $x \le n$ 为止。结果，当 $N \to \infty$ 时，对于 $1 \le n \le N$ 和固定的 m，固定元素的平均数趋于 $\sum_{k \ge 1}(m-1)^k/(m^{k+1}-(m-1))^k$。[由于这个值位于 $(m-1)/m$ 和1之间，因此Josephus排列比起随机排列来，固定元素稍微少些。]

32. [*M25*] (a)证明形如

$$\pi = (2\ 3)^{e_2}(4\ 5)^{e_4}\cdots(2m\ 2m+1)^{e_{2m}}(1\ 2)^{e_1}(3\ 4)^{e_3}\cdots(2m-1\ 2m)^{e_{2m-1}}$$

的任何排列 $\pi=\pi_1\pi_2\cdots\pi_{2m+1}$，其中每个 e_k 为0或1，对于 $1 \le k \le 2m+1$，有 $|\pi_k - k| \le 2$

(b)给定 $\{1,2,\cdots,n\}$ 的任何排列 ρ，试构造上述形式的一个排列 π，使得 $\rho\pi$ 是一个单循环因此每个排列"近乎"一个循环。

33. [*M33*] 如果 $m=2^{2^l}$ 和 $n=2^{2l+1}$，试说明对于 $0 \le j < m$ 如何构造排列 $(\alpha_{j1},\alpha_{j2},\cdots,\alpha_{jn};\beta_{j1},\beta_{j2},\cdots,\beta_{jn})$ 的序列，要求它们具有以下的"正交"性：

$$\alpha_{i1}\beta_{j1}\alpha_{i2}\beta_{j2}\cdots\alpha_{in}\beta_{jn} = \begin{cases}(1\ 2\ 3\ 4\ 5), & \text{如果 } i=j\\ (), & \text{如果 } i \ne j\end{cases}$$

每个 α_{jk} 和 β_{jk} 应是 $\{1,2,3,4,5\}$ 的一个排列。

▶ 34. [*M25*]（转置数据块）实践中需要的最普通的排列之一是把 $\alpha\beta$ 变成 $\beta\alpha$，其中 α 和 β 是一个数组的子串。换言之，如果 $x_0x_1\cdots x_{m-1}=\alpha$ 和 $x_mx_{m+1}\cdots x_{m+n-1}=\beta$，我们要把数组 $x_0x_1\cdots x_{m+n-1}=\alpha\beta$ 改变成数组 $x_mx_{m+1}\cdots x_{m+n-1}x_0x_1\cdots x_{m-1}=\beta\alpha$；对于 $0 \le k < m+n$，每个元素 x_k 都应被换成 $x_{p(k)}$，其中 $p(k)=(k+m) \bmod (m+n)$。试证明每个这样的"循环移位"排列有一个简单的循环结构，并利用这一结构设计对于所求的重新安排的一个简单算法。

35. [*M30*] 继续上道题，令 $x_0x_1\cdots x_{l+m+n-1}=\alpha\beta\gamma$，其中 α，β 和 γ 是长度分别为 l，m 和 n 的串，并假设我们要把 $\alpha\beta\gamma$ 改变成 $\gamma\beta\alpha$。试证明相应的排列有一个导致有效算法的方便的循环结构。[习题34考虑的是 $m=0$ 的特殊情况。]提示：考虑把 $(\alpha\beta)(\gamma\beta)$ 改变成 $(\gamma\beta)(\alpha\beta)$。

36. [*27*] 写出习题35答案中算法的一个MIX子程序，并分析其运行时间。试把它同将 $\alpha\beta\gamma$ 变成 $(\alpha\beta\gamma)^R=\gamma^R\beta^R\alpha^R$ 再变成 $\gamma\beta\alpha$ 的一个更简单方法作比较，其中 σ^R 表示串 σ 的左右颠倒。

1.4　某些基本程序设计技术

1.4.1　子程序

当在一个程序中的若干个不同的地方，都要执行同一个确定的任务时，通常不希望在每个地方都重复编码。为避免这种重复的情况，这种编码(称为子程序)可以只放在一个地方，而且在这个子程序完成之后，可以附加少量额外的指令，以便适当重新开始外部的程序。子程序与主程序之间控制的转移，称为子程序的链接。

每部机器都有它自己实现子程序链接的特殊方式，通常这会涉及一些特殊的指令。在 MIX 中，J 寄存器就用于这一目的；我们的讨论就将以 MIX 机器语言为基础，但类似的说明也适用于其它计算机的子程序链接。

子程序用来节省一个程序的空间；但它并不节省任何时间，除了因为通过使用较少的空间而不明显地节省了时间外——例如，使用较少的时间装入程序，或者在程序中只须较少的扫描遍数，或者在配有若干级存储器的机器上更好地使用高速存储器。进入和离开子程序的额外时间通常是可以忽略不计的。

子程序还有许多其它优点。它们使得想像一个庞大而又复杂的程序的结构变得较为容易些；它们形成整个问题的一个逻辑段，这通常使得对程序的调试也变得较为容易。由于子程序可以为非子程序编制者所用，因而子程序还有额外的价值。

大多数计算机装置都配备了有用子程序的大型的库，这样的库大大地方便了标准计算机应用的程序设计。然而，一个程序员不应该把这想像为子程序的惟一目的；子程序不应该总是被当做为公众使用的“通用”程序。专用子程序也一样重要，即使只打算让它们出现在一个程序中。1.4.3 小节包含有若干个典型的例子。

最简单的子程序是只有一个入口和一个出口的子程序，例如我们已经讨论过的 MAXIMUM 子程序(参见 1.3.2 小节，程序 M)。为便于参考，这里我们重新转抄这个程序，但是略作改动，使其检索固定的单元数 100，以寻找极大值。

```
 * MAXIMUM OF X[1..100]
MAX100   STJ    EXIT   子程序链接
         ENT3   100    M1. 初始化
         JMP    2F
1H       CMPA   X,3    M3. 比较
         JGE    *+3
2H       ENT2   0,3    M4. 改变 m                    (1)
         LDA    X,3    找到新的极大值
         DEC3   1      M5. k 减 1
         J3P    1B     M2. 全部测试过了吗?
EXIT     JMP    *      返回主程序 ▌
```

在以这段代码为子程序的更大程序中，一条指令“JMP MAX100”将引起寄存器 A 被置成为从单元 X + 1 到单元 X + 100 的当前极大值，而且极大值的位置将出现于 rI2 中。在这种情况下子程序链接是通过指令“MAX100 STJ EXIT”和后边的“EXIT JMP *”实现的。由于 J 寄存器的运作方式，出口指令将跳到原来对 MAX100 进行访问的那个位置的下一个单元。

较新的计算机，例如注定要代替 MIX 的机器 MMIX，有记住返回地址的更好的方式。主要差别在于，在内存中程序指令不再被修改；相关信息被保存在寄存器中或者在一个特殊的数组中，而不是在程序本身内。（参见习题 7。）本书的下一版将采用这个现代的观点，但现在我们还仍将坚持自修改代码的老做法。

不难得到当使用子程序时所节省的代码数量和失去的时间数量的定量的论述。假设一块代码需要 k 个单元而它在程序中出现了 m 处。如果把这重写成为一个子程序，我们需要一条额外的指令 STJ 和这个子程序的出口行，另外在调用这个子程序的 m 处地方的每个地方，加上一条 JMP 指令，这就给出总共 $m + k + 2$ 个单元，而不是 mk，所以所节省的数量是

$$(m - 1)(k - 1) - 3 \tag{2}$$

如果 k 为 1 或者 m 为 1，那通过使用子程序我们可能不能节省任何空间；当然，这是明显的。如果 k 为 2，为获得好处，m 必须大于 4，等等。

所损失的时间数量是额外的指令 JMP，STJ 和 JMP 所花费的时间，如果不使用这个子程序它们就不存在；因此，如果在一个程序的运行期间使用这个子程序 t 次，那就需要 $4t$ 的额外时间。

对这些估计切勿全信，因为它们是对一种理想化的情况给出的。许多子程序不可能只通过一条指令 JMP 来调用。而且，如果不使用子程序的方法，而是在一个程序的许多地方重复编写代码，则对于每部分的代码还可能利用它所处的特定环境予以优化。而另一方面，为使用子程序，代码必须按最一般而不是特定的情况来编写，因此这通常将添加若干额外的指令。

当把子程序写成处理一般的情况时，它借助于*参数*来表达。参数是支配子程序的动作的值；从一次子程序调用到另一次调用，这些参数的值将随之改变。

在外部程序中，把控制转移到子程序并使它适当地开始的代码，称做*调用序列*。调用子程序时提供的参数的特定值，称做*变元*。对于我们的 MAX100 子程序，调用序列很简单，就是“JMP MAX100”。但是当必须提供变元时，一般需要较长的调用序列。例如，程序 1.3.2M 是 MAX100 的一个推广，它求表的头 n 个元素的极大值。参数 n 出现在变址寄存器 1 中，而且它的调用序列

```
LD1   =n=              ENT1   n
JMP   MAXIMUM    或    JMP    MAXIMUM
```

涉及两个步骤。

如果调用序列使用 c 个内存单元，关于所节省空间量的公式(2)变成

$$(m - 1)(k - c) - 常数 \tag{3}$$

而且用于子程序链接所损失的时间会稍微增加。

由于某些寄存器可能需要予以保留和恢复,因此对于上边的那些公式可能需要作进一步的校正。例如,在 MAX100 子程序中,我们必须记住,通过写“JMP MAX100”我们不仅仅获得在寄存器 A 中的极大值,以及在寄存器 I2 中获得它的位置;而且我们还置寄存器 I3 为零。一个子程序可能会破坏寄存器的内容,这一点必须记住。为了防止 MAX100 改变 rI3 的设置,就需要包括另外的指令。对 MIX 而言做这件事最短也最快的方法是在 MAX100 之后插入指令“ST3 3F(0:2)”,以及在 EXIT 之前插入“3H ENT3 *”。净代价将是额外的两行代码加上在每次调用子程序时的三个机器周期。

一个子程序可以当做计算机机器语言的一个*扩充*。通过在内存中有一个 MAX100 子程序,我们现在便有了作为极大值寻找程序的单个指令(即“JMP MAX100”)。重要的是,要仔细定义每个子程序的作用,一如定义机器语言操作符本身那样;因此,程序员必须确保写出每个子程序的特征,即便没有别的人将利用该子程序或其特征描述也一样。在 1.3.2 小节给出的 MAXIMUM 的特征如下:

$$\left.\begin{array}{ll}\text{调用序列:} & \text{JMP MAXIMUM}\\ \text{入口条件:} & \text{rI1} = n;\ \text{假定}\ n \geq 1\\ \text{出口条件:} & \text{rA} = \max\limits_{1\leq k\leq n}\text{CONTENTS}(\text{X}+k) = \text{CONTENTS}(\text{X}+\text{rI2})\\ & \text{rI3} = 0;\ \text{rJ 和 CI 也受影响}\end{array}\right\}\qquad(4)$$

(习惯上我们将略去不谈寄存器 J 和比较指示器也受子程序影响这一事实;这里提及它们仅仅为了完备起见。)注意 rX 和 rI1 不受子程序动作的影响,否则在出口条件处就会提及这些寄存器了。在特征描述中还须说明子程序外部可能会受影响的所有内存单元;对子程序 MAXIMUM 而言,由其特征描述可知,没有存什么东西,因为(4)中没有提到关于存储器的改变。

现在让我们考虑对子程序的多个入口。假设有需要通用子程序 MAXIMUM 的一个程序,但它通常要使用其中 $n = 100$ 的特殊情况 MAX100,这两者可以组合如下:

```
MAX100    ENT3    100     头一个入口
MAXN      STJ     EXIT    第二个入口                 (5)
          JMP     2F      像在(1)中那样继续
...
EXIT      JMP     *       返回到主程序  ▌
```

子程序(5)实际上和(1)一样,只是头两条指令交换;我们使用了这样一个事实,即“ENT3”不改变 J 寄存器的设置。如果我们要对这个子程序加入第三个入口 MAX50,我们可以于开始处插入代码

```
MAX50     ENT3    50
          JSJ     MAXN                               (6)
```

(回想一下“JSJ”意味着转移而不改变寄存器 J 的内容。)

当参数的个数不多时,通常希望通过把参数放进方便的寄存器中(如同我们在 MAXN 中使用 rI3 保存参数 n 及在 MAXIMUM 中使用 rI1 保存参数 n 一样),而把它们传送

到一个子程序中，或者通过把它们存进固定的存储单元来传送。

另一个提供变元的方便方法是简单地把变元列在 JMP 指令之后，子程序可以访问到它的参数，因为它知道 J 寄存器的设置。例如，如果要使对 MAXN 的调用序列成为

```
        JMP     MAXN
        CON     n                                           (7)
```

则子程序可以写成如下：

```
MAXN    STJ     *+1
        ENT1    *          rI1←rJ
        LD3     0,1        rI3←n
        JMP     2F         像在(1)中那样继续                  (8)
...
        J3P     1B
        JMP     1,1        返回  ▌
```

在像 System/360 那样的机器上，链接通常是通过把出口单元放进一个变址寄存器来实现的，这么做是特别方便的。当一个子程序需要许多变元时，或者当一个程序已由一个编译程序写成时，它也是有用的。然而，在这个情况下上面所用的多入口技术经常失败。我们可以通过写

```
MAX100  STJ     1F
        JMP     MAXN
        CON     100
1H      JMP     *
```

来“冒充它”，但这已不像(5)那样有吸引力了。

类似于把变元列在转移指令之后的技术通常用于有多个出口的子程序。多个出口意味着我们要这个子程序依赖于它所探测的条件返回若干不同单元之一。在最严格的意义下，一个子程序出口的单元是一个参数；所以，如果依赖于不同情况，有它可以出口的若干位置，它们应当作为变元来加以提供。我们的“极大值”子程序的最后例子将有两个入口和两个出口。调用序列是

对于一般的 n	对于 $n=100$
ENT3 n	
JMP MAXN	JMP MAX100
如果 max≤0 或 max≥rX 出口于此	如果 max≤0 或 max≥rX 出口于此
如果 0<max<rX 出口于此	如果 0<max<rX 出口于此

（换言之，当极大值为正和小于寄存器 X 的内容时，使出口位于转移指令后面两个单元处。）对这些条件的子程序可容易地写出：

```
MAX100    ENT3    100      对于 n=100 的入口
MAXN      STJ     EXIT     对于一般的 n 的入口
          JMP     2F       像在(1)中那样继续
```

```
…
            J3P     1B
            JANP    EXIT    如果 max≤0 则走通常的出口              (9)
            STX     TEMP
            CMPA    TEMP
            JGE     EXIT    如果 max≥rX 则走通常的出口
            ENT3    1       否则走第二个出口
EXIT        JMP     *,3     返回到适当位置 ▌
```

子程序可以调用其它子程序；在复杂的程序中，有五层以上嵌套的子程序调用并非不寻常。当使用这里所述的链接时必须遵守的惟一限制是，子程序不能对正在调用（包括直接或间接）它的任何子程序进行调用。例如，考虑下面的情况

```
[主程序]  |    [子程序 A]        |    [子程序 B]        |    [子程序 C]
          | A        STJ EXITA   | B        STJ EXITB   | C        STJ EXITC
   ⋮      |              ⋮       |              ⋮       |              ⋮          (10)
 JMP A    |          JMP   B     |          JMP   C     |          JMP   A
   ⋮      |              ⋮       |              ⋮       |              ⋮
          |  EXITA   JMP   *     |  EXITB   JMP   *     |  EXITC   JMP   *
```

如果主程序调用 A，而 A 调用 B，B 调用 C，然后 C 调用 A，则访问主程序的 EXITA 中的地址被破坏，因此无法返回到该程序。类似的说明也适用于被每一个子程序使用的全部暂存单元和寄存器。能适当地处理这种递归情况的子程序链接并不难设计，第 8 章将详细讨论递归问题。

我们通过简单地讨论怎样写出一个复杂的和冗长的程序，来作为这一节的结束。我们如何决定将需要什么类型的子程序，以及应当使用什么调用序列？为确定这些问题，有一个成功的方法，这就是使用一套迭代步骤：

步骤 0.（初始想法）　首先我们笼统地决定着手这个程序的总计划。

步骤 1.（程序的一个粗略草案）　接着我们以任何方便的语言写出程序的“外层”。E. W. Dijkstra，*Structured Programming*（Academic Press，1972），第 1 章和 N. Wirth，*CACM* **14**（1971），211～227，非常精辟地叙述了关于进行这项工作的一个颇为系统的方法。我们可以通过把整个程序分成一些小块开始，把这些小块想像作为子程序，尽管它们仅仅被调用一次。这些块又逐次地细化成越来越小的部分，相应地这些小的部分有更简便的任务。每当出现某个计算任务，而它似乎可能会在别处出现或者已经在别处出现过，则我们就可以定义一个（真实的一个）完成这项工作的子程序。这时我们还不写出这个子程序来。我们假定这个子程序已经执行了它的任务，而继续来写主程序。最后，当主程序已经被勾画出来之后，我们才回过头来着手写子程序。首先，要试图拿下最复杂的子程序以及它们的子子程序，等等。用这样的方法，我们就得到了一张子程序清单。每个子程序的实际功能大概已经改动了若干次，以致我们的草案的最初部分如今已不正确了。但这并不成问题，因为它只不过是一个草案。对于每个子程序，关于如何调用它以

及它的通用性如何,我们现在已经有了一个相当好的想法了。通常,值得使每个子程序的通用性稍微更广些。

步骤 2.(头一个工作程序) 这一步骤以相反于步骤 1 的方向进行。我们现在用计算机算法语言,比如说用 MIXAL 或 PL/MIX 或者一种高级语言来写;这一次,我们由最低层的子程序开始,并且最后才来写主程序。只要可能,我们在一个子程序本身已经编出之前,绝不试图去写任何调用这个子程序的指令。(在步骤 1,我们尝试做相反的工作,即在写出所有对它的调用之前,绝不去考虑这个子程序。)

在这个过程中,随着越来越多的子程序被编写出来,我们的自信心也逐渐增强,因为我们正继续地扩充正在为之进行程序设计的机器的功能。在编出一个独立的子程序之后,我们就要立即准备一个关于它干什么工作,以及它的调用序列是什么等等的完整描述,如同在(4)中那样。不要使临时存储单元相重叠,这一点也很重要;如果每个子程序都访问单元 TEMP,那可能将是极大的灾难,尽管在步骤 1 编制程序的草案时,不去为这样的问题操心是合宜的。克服重叠忧虑的一个明显的方法,是使每个子程序都使用它自己的临时存储单元;但如果这太浪费空间,则另一个相当好的方案,是命名单元 TEMP1,TEMP2,等等;在一个子程序内编号从 TEMPj 开始,其中 j 比这个子程序的任何子子程序所用的最大的数大 1。

步骤 3.(重新检查) 步骤 2 的结果应该是非常接近于一个工作程序了,但是也许还有可能改进。为此,一个好办法就是再次以相反的方向进行,对于每个子程序,研究对它所做的所有调用。应该把子程序扩大成来做某些更为普遍的事情,即在使用这个子程序之前或之后总由外部的程序来做的事情,也许会更好些。或许应当把若干个子程序合并为一;或许一个子程序仅被调用过一次,因而根本就不应把它作为一个子程序。(或许对一个子程序连一次调用也没有,因此可以整个去掉。)

这时,废弃每件事情而从步骤 1 重新开始通常是一个好的想法!这并不是开玩笑;到这一步为止所花费的时间并不是被浪费了,因为关于这个问题我们已经学到了很多。通过事后的反思,我们可能已经发现对于程序的整体组织能做的若干改进。没有理由为返回步骤 1 而感到害怕——它比再由步骤 2 进行到步骤 3 容易得多,因为一个类似的程序已经完成了。而且,我们很可能将节省同重写程序一样多的查错时间。某些已经写成的最好的计算机程序,其成功都归结于这样一个事实:在这个阶段前后的所有工作全都在无意中推倒重来,因而编写者须从头开始。

另一方面,大概从没有一个复杂的计算机程序能做到再也不须改进了,因此步骤 1 和 2 不应无限次地重复。当显然地可以做重要的改进时,那就很值得再花额外的时间重新开始,但是,最终是会达到不再反复的境界的。

步骤 4.(调试) 在程序经过最后的推敲,包括存储分配以及其它收尾细节的推敲之后,就该是从另一个方向对它进行考察的时候了,这与步骤 1,2 和 3 这三者所沿的方向不同——现在我们按照计算机执行程序的顺序来研究程序。当然,这可以用手工,或者通过机器来做。作者感到,在这时,利用系统例行程序,让例行程序跟踪每条指令的头两次执行,是十分有帮助的;重要的是,要重新考虑那些奠定程序基础的思想并验证发生的每件事情,是否恰如所预期的那样。

调试是一门技巧,需要进行许多进一步的研究,而且实现调试的途径,高度地依赖于每个计算机装置上可利用的设施。进行有效调试的一个良好开端通常是准备恰当的测试数据。最有效的调试技术,似乎就是被设计和被构造在程序本身之中的那些——今天许多最好的程序员会牺牲近一半的程序,来方便另一半程序的调试过程;这头一半通常是由相当直截了当的、以便于阅读的格式显示有关信息的例行程序组成。它将最终被抛弃,但是其净结果是在生产效率上令人震惊的收获。

另一个好的调试做法,是对每个错误保留一份记录。尽管这大概将是十分麻烦的,但这样的信息对于进行调试问题研究的任何人都是无价之宝,而且它还将帮助你学习怎样来减少未来的错误。

注:作者在成功地完成了许多中等规模的软件项目之后,但在形成成熟的程序设计风格之前,于 1964 年写下了前边论述的大部分。后来,在 20 世纪 80 年代,作者认识到,称为结构化文档或文字化(literate)程序设计的另外一种技术,可能更为重要。*Literate Programming* (Cambridge Univ. Press, 1992 年初版)一书中总结了作者对编写各种程序最佳方法的当前认识。该书的第 11 章包含了在 1978—1991 年期间从 $\TeX$ 程序中消除的所有错误的详细记录。

Up to a point it is better to let the snags [bugs] be there
than to spend such time in design that there are none.
(how many decades would this course take?)
直到此时,最好就让错误暂时就留在那里
而不是在设计阶段花费那么多时间使错误不存在。
(掌握这一点要花费多少个十年?)
—A.M.TURING, *Proposals for ACE* (1945)

习　题

1. [*10*] 如同(4)给出子程序 1.3.2M 的特征那样,指出子程序(5)的特征。

2. [*10*] 试不用 JSJ 指令,而给出替代(6)的代码。

3. [*M15*] 将(4)中的情况叙述完备,指出作为子程序执行的结果,寄存器 J 和比较指示器发生了什么情况;还指出,如果寄存器 I1 不是正的,则将发生什么情况。

▶ 4. [*21*] 通过寻求 X[a],X[$a+r$],X[$a+2r$],…,X[n]的极大值,其中 r 和 n 是参数,而且 a 是满足 $a\equiv n$(modulo r)的最小正数,即 $a=1+(n-1)\bmod r$,试写出推广 MAXN 的一个子程序。对于 $r=1$ 的情况给出一个特殊入口。像在(4)中那样,列出你的程序的特征。

5. [*21*] 假设 MIX 没有 J 寄存器。试设计不使用寄存器 J 的用于子程序链接的一个手段,并通过把 MAX100 子程序写成在效果上等价于(1),来给出你的设计的一个例子。以类似于(4)的方式,指出这个子程序的特征(保留 MIX 的自修改代码约定)。

▶ 6. [*26*] 假设 MIX 没有 MOVE 操作符,试写出标题为 MOVE 的一个子程序,使得调用序列"JMP MOVE; NOP A,I(F)"具有和"MOVE A,I(F)"一样的效果,如果后者允许的话。惟一的差别应该是对寄存器 J 的影响,以及事实上一个子程序自然地要比一个硬件指令消耗更多时间和空间。

▶ 7. [*20*] 为什么自修改代码现在不受欢迎?

1.4.2 共行程序

子程序是更一般的称为共行程序(coroutines)的程序组成部分的特殊情况。和一个主程序与一个子程序之间非对称的关系相对照,在共行程序之间有一个完全对称的关系,它们彼此调用。

为了理解共行程序的概念,让我们考虑有关子程序的另 一种思路。在上一小节中所持的观点是,一个子程序仅仅是被引入来节省程序行数的计算机硬件的一个扩充。这可能是真的,但也可以有另外一种观点:我们可以把主程序和子程序当做一个程序班子,班子的每个成员都有某种工作要做。在进行它自己的工作的过程中,主程序将激活子程序;子程序将执行它自己的功能并且然后激活主程序。我们可以拓展我们的想像而相信,从子程序的观点来说,当它出口时它调用主程序;主程序继续执行它的任务,然后"出口"到子程序。子程序运行,而后再调用主程序。

这个稍微牵强的原理实际上适用于共行程序,对共行程序而言,不可能区分哪一个是另外一个的子程序。假设我们有共行程序 A 和 B;当进行 A 的程序设计时,我们可以把 B 想像为子程序;但当对 B 进行程序设计时,我们又可把 A 想像为子程序。即在共行程序 A 中,指令"JMP B"用来激活 B;在共行程序 B 中指令"JMP A"用来再次激活 A。每当一个共行程序被激活,它就在它上次的动作被搁置处继续执行它的程序。

例如,共行程序 A 和 B 可以是下棋的两个程序。我们可以把它们组合在一起使得它们彼此针对对方来对弈。

通过 MIX,共行程序 A 和 B 之间这样的链接是通过在程序中包括以下四条指令完成的:

```
A    STJ  BX          B    STJ  AX
                                        (1)
AX   JMP  A1          BX   JMP  B1
```

这要求对于每个方向的控制转移使用四个机器的周期。开始时,AX 和 BX 被设置成转移到每个共行程序的起始地址 A1 和 B1。假设我们首先在单元 A1 处启动共行程序 A。当它从单元 A2 处执行"JMP B"时,在单元 B 中的指令把 rJ 存入 AX 中,然后 AX 的内容为"JMP A2 + 1"。BX 中的指令使我们达到单元 B1,在共行程序 B 开始它的执行后,最终它将达到在比如说单元 B2 处的指令"JMP A"。我们保存 rJ 于 BX 中并转移到单元 A2 + 1 处,继续执行共行程序 A 直到它再次转移到 B,它把 rJ 存于 AX 中并转移到 B2 + 1,等等。

程序—子程序和共行程序—共行程序链接之间的实质性差别,如同通过研究上边的例子所看到的,在于一个子程序总是在它的开始处被启动,它通常是一个固定的位置;主程序或共行程序总是在它上次结束处的下一位置被启动。

共行程序在实践中最自然地出现于当它们同输入和输出的算法相关联时。例如,假设共行程序 A 的任务是读卡片并且在输入时实现某种转换,并把它简化成一系列的项。另一个共行程序,我们将称之为 B,它进一步处理这些项,并打印答案。B 将周期

地调用由 A 找到的逐次的输入项。因此,每当共行程序 B 需要下一个输入项时,它就转移到 A 去,而每当一个输入项已经被找到时,共行程序 A 转移到 B。读者可能说:“好,B 是主程序而 A 仅仅是进行输入的一个子程序。”然而,当过程 A 非常复杂的时候,这就变得不太对了;其实,我们可以把 A 想像为主程序而把 B 当做进行输出的一个子程序,而且上述的说明仍然正确。共行程序思想的有用性正出现于这两个极端的中间,此时 A 和 B 两者都复杂而且每个都在多处位置调用另一个。要找出说明共行程序这一思想的重要性的简短例子是相当困难的,最有用的共行程序应用一般都十分长。

为了研究共行程序的动作,让我们来考虑一个凑成的例子。假设我们要写把一个代码翻译成另一个代码的程序。要被翻译的输入代码是以一个句点结束的字符序列,例如

A2B5E3426FG0ZYW3210PQ89R.　　(2)

这已被穿孔在卡片上;对于出现于这些卡片上的空白列,予以忽略。从左到右,对输入的理解如下:如果下一字符是一个数字 0,1,…,9 比如说 n,它表示 $n+1$ 次重复下一个字符,无论下一个字符是否数字。一个非数字简单地表示它自己。我们程序的输出是由以这个方式表示的序列组成的,而且每三个字符分为一组,直到出现一个句点为止;最后一组可能少于三个字符。例如,(2)应由我们的程序翻译成为

ABB BEE EEE E44 446 66F GZY W22 220 0PQ 999 999 999 R.　　(3)

注意 3426F 并不意味着 3427 次重复 F;它表示着四个 4 和三个 6,后边跟着是 F。如果输入序列是“1.”,则输出只是“.”,而不是“..”,因为第一个句点结束输出。我们的程序应把输出穿孔到卡片上,而且在每张卡片上,除了可能的最后一个例外外,三个一组,共有十六个组。

为了实现这一翻译,我们将写两个共行程序和一个子程序。称做 NEXTCHAR 的子程序被设计成寻找输入的非空字符,而且把下一个这样的字符放进寄存器 A 中:

```
*   SUBROUTINE  FOR  CHARACTER  INPUT
READER     EQU    16               卡片输入机的设备号
INPUT      ORIG   *+16             输入卡片的位置
NEXTCHAR   STJ    9F               子程序入口
           JXNZ   3F               开始时 rX = 0
1H         J6N    2F               开始时 rI6 = 0
           IN     INPUT(READER)    读下一张卡片
           JBUS   *(READER)        等候完成
           ENN6   16               令 rI6 指向头一个字
2H         LDX    INPUT+16,6       取输入的下一个字
           INC6   1                推进指针
3H         ENTA   0
           SLAX   1                下一字符→rA
9H         JANZ   *                跳过空白
```

```
                JMP   NEXTCHAR + 1
```

这个子程序有下列特征：

调用序列：JMP NEXTCHAR。

入口条件：rX = 有待使用用的字符，rI6 指向下一个字，或

rI6 = 0 表示必须输入新的卡片。

出口条件：rA = 输入的下一非空字符；把 rX 和 rI6 置为 NEXTCHAR 的下个入口。

我们的头一个共行程序称为 IN，它寻找要适当重复的输入代码的字符。初始时，它在单元 IN1 开始：

```
*   FIRST   COROUTINE
2H          INCA   30           找到的非数字
            JMP    OUT          把它发送到 OUT 共行程序
IN1         JMP    NEXTCHAR     获得字符
            DECA   30
            JAN    2B           它是一个字母吗？
            CMPA   =10=
            JGE    2B           它是一个特殊字符吗？
            STA    *+1(0:2)     找到数字 n
            ENT5   *            rI5←n
            JMP    NEXTCHAR     获得下一字符
            JMP    OUT          把它发送到 OUT 共行程序
            DEC5   1            n 减 1
            J5NN   *-2          如果必要则重复
            JMP    IN1          开始新的循环
```

（回想一下，在 MIX 的字符代码中，数字 0 ~ 9 有代码 30 ~ 39）。这个共行程序有下列特征：

调用序列：JMP IN。

出口条件（当转移到 OUT 时）：rA = 下个输入字符且有适当的重复；

rI4 不改变进入时的值。

入口条件（返回时）：rA，rX，rI5，rI6 应当不改变它们在上一次出口时的值。

称为 OUT 的另一个共行程序，把代码分成为三个字符一组并且穿孔到卡片上，初始时它在 OUT1 处开始：

```
*   SECOND   COROUTINE
            ALF                 用作空白的常数
OUTPUT      ORIG   *+16         供答案用的缓冲区域
PUNCH       EQU    17           穿孔卡片的设备号
OUT1        ENT4   -16          开始新输出卡
            ENT1   OUTPUT
```

```
            MOVE   -1,1(16)          置输出区域为空白
1H          JMP    IN                获得下一个被翻译的字符
            STA    OUTPUT+16,4(1:1)  把它存于(1:1)字段中
            CMPA   PERIOD            它是"."吗?
            JE     9F
            JMP    IN
            STA    OUTPUT+16,4(2:2)  把它存于(2:2)字段中
            CMPA   PERIOD            它是"."吗?
            JE     9F
            JMP    IN                若不是,取另一个字符
            STA    OUTPUT+16,4(3:3)  把它存于(3:3)字段中
            CMPA   PERIOD            它是"."吗
            JE     9F
            INC4   1                 移至输出缓冲区中的下一个字
            J4N    1B                卡片结束了吗?
9H          OUT    OUTPUT(PUNCH)     若是,把它穿孔
            JBUS   *(PUNCH)          等候完成
            JNE    OUT1              除非已经发现".",否则为了更
            HLT                        多字符而返回
PERIOD      ALF    ␣␣␣␣.
```

这个共行程序有下列特征：

调用序列：JMP　OUT。

出口条件(当转移到 IN 时)：rA,rX,rI5,rI6 不改变入口时它们的值;rI1 可能受影响;在输出中记录前一个字符。

入口条件(当返回时)：rA = 输入的下一字符且有适当的重复;rI4 不改变上一次出口时它的值。

为了完成这个程序,我们还需要写共行程序链接(参见(1))和适当初始化。共行程序的初始化有点技巧,尽管实际上并不困难。

```
*  INITIALIZATION  AND  LINKAGE
START       ENT6   0                 为 NEXTCHAR 初始化 rI6
            ENTX   0                 为 NEXTCHAR 初始化 rX
            JMP    OUT1              以 OUT 开始(参见习题 2)
OUT         STJ    INX               共行程序链接
OUTX        JMP    OUT1
IN          STJ    OUTX
INX         JMP    IN1
            END    START
```

这就完成了这个程序。读者应仔细研究它,特别要注意怎样独立地编写每个共行程序,而想像另一个共行程序为它的子程序。

IN 和 OUT 共行程序的入口和出口条件在上述程序中完美地啮合。一般说来,我们并不都是这样幸运,而且共行程序链接也还将包括装入和存储适当的寄存器的指令。例如,如果 OUT 将破坏寄存器 A 的内容,则共行程序链接将变成

```
OUT     STJ   INX
        STA   HOLDA        当离开 IN 时保存 A
OUTX    JMP   OUT1                                  (4)
IN      STJ   OUTX
        LDA   HOLDA        当离开 OUT 时恢复 A
INX     JMP   IN1          ▌
```

在共行程序和多遍(趟)算法(multiple-pass algorithm)之间存在一个重要的关系。例如,我们刚刚描述过的翻译过程可以在两次不同的扫描中完成。我们可以首先仅仅完成 IN 共行程序,应用它于整个输入过程,并且把每个字符以及它的适当数量的重复写到磁带上。在这遍完成之后,我们可以把带重绕而后只执行 OUT 共行程序,从带上三个一组的字符序列中取字符。这便叫做"两遍"过程。(直观上说,一"遍"指输入的一个完整的扫描。这个定义并非精确,而且在许多算法中所取的遍数是全然不清楚的;但尽管它有含糊性,"遍"的直观概念是有用的。)

图 22(a)图解一个四遍的过程,我们将十分经常地发现,如同在图 22(b)中所示那样,同样的过程可以在仅仅一次的扫描中完成,如果我们以四个共行程序 A,B,C,D 代替分别的扫描 A,B,C,D 的话。当遍 A 已经把输入的一项写到带 1 上时,共行程序 A 将转移到 B;当遍 B 已经从带 1 读入输入的一项时共行程序 B 将转移到 A。而当 B 已经把输出的一项写到带 2 上时,B 将转移到 C;等等。UNIX® 的用户将把这识别为"管道",并以"Pass A| Pass B| Pass C| Pass D"表示。对于遍 B,C 和 D 的程序有时称做"过滤器"。

反过来,一个由 n 个共行程序完成的过程通常可以被转换为一个 n 遍过程。由于这个对应,值得来把多遍算法和一遍算法做一比较。

a)心理上的区别　对同一问题,多遍扫描的算法一般要比一遍扫描的算法更易于建立和理解。把一个过程分成为一个挨一个地相继进行的一些小步骤的序列,比起许多转换同时进行的一个复杂过程来,是更容易理解的。

还有,如果所处理的是一个很大的问题,而且许多人合作来编制一个计算机程序,则多遍算法提供了把工作进行分工的自然的方式。

一个多遍扫描算法的这些优点在共行程序中也存在,因为每个共行程序实际上可以独立于其它共行程序而写出,而且链接使得表面上是多遍扫描的算法成为一个单遍扫描的过程。

b)时间上的区别　为组装、写、读以及卸装在诸遍之间流动的中间数据所需要的时间(例如,在图 22 的诸带上的信息)在一遍扫描算法中可被避免。由于这个原因,一遍扫描算法将是更快的。

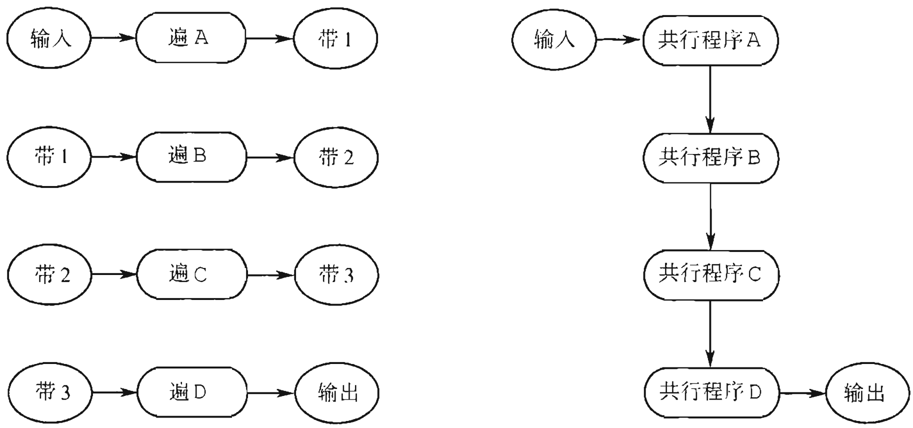


图 22　遍数:(a)一个四遍算法及(b)一遍算法

c)空间上的区别　一遍扫描算法需要空间同时保存所有程序于内存中,而多遍扫描算法一次仅要求保存一次扫描的程序所需要的内存空间。这个要求可能影响速度,甚至有比 b)中所指出的还要大的影响。例如,许多计算机都有一个有限数量的“快速内存”和更大数量的较慢的内存;如果每遍扫描仅仅刚好放进快速内存中,其结果将比我们在单遍扫描中使用共行程序快得多(因为使用共行程序大概将迫使大多数程序放在较慢的内存中,否则就得反复地在快速内存中转入或转出)。

有时有必要一次就设计出供若干种计算机结构使用的算法,这些计算机结构中有的比其它有更大的内存容量。在这样的情况下,有可能借助于共行程序来编写程序,而且使内存大小支配扫描的遍数:把尽可能多的共行程序装入在一起,而且为丢失的链接提供输入或输出的子程序。

尽管在共行程序和遍数之间的关系是重要的,但我们应当记住,共行程序的应用并不总是被分开成多遍扫描算法。如果共行程序 B 从 A 获得输入而且还向 A 发回关键信息,如同早先提到的下棋例子中那样,则动作序列不能在转换成遍 A 之后就接着做遍 B。

反过来,显然某些多遍扫描算法不能被转换成共行程序。某些算法固有地是多遍扫描的;例如,第二遍扫描可能需要头一遍扫描累计的信息(比如在输入中某个字出现的总次数),在这方面有一个老笑话值得一提:

一位瘦小的老太太,乘坐公共汽车,她问:“小孩,请你告诉我到帕萨迪纳大街在哪儿下车?”

小孩说:“你只要看着我,并且在我下车的头两站下车。”

(这个笑话说的是,这个小孩给出了一个两遍扫描的算法。)

关于多遍扫描的算法就谈这些。我们将在本书中的许多地方见到关于共行程序的更多例子,例如,像在 1.4.4 小节中的缓冲方案部分那样。共行程序在离散系统模拟中也起着重要作用,请参见 2.2.5 小节。在第 8 章中要讨论重复的共行程序的重要思想,

其某些有趣应用可在第 10 章中找到。

习 题

1. [*10*] 说明为什么一本教科书的作者难以举出共行程序的简短例子。

▶ 2. [*20*] 正文中的程序首先启动 OUT 共行程序。如果首先执行的是 IN 程序，则将发生什么情况？即如果把行 60 的"JMP OUT1"改变成为"JMP IN1"，情况会怎样？

3. [*20*] 真或假：在 OUT 内的三条"CMPA PERIOD"指令都可以被省略，而这个程序仍然有效。（仔细地观察。）

4. [*20*] 说明类似于(1)的共行程序链接对于你所熟悉的真实的计算机如何给出？

5. [*15*] 假设两个共行程序 IN 和 OUT 需要寄存器 A 的内容来保留出口和入口之间的不受触动；换言之，假设每当指令"JMP IN"出现于 OUT 之内，则当控制返回到下一行时寄存器 A 的内容不变，而且关于在 IN 之内的"JMP OUT"也作一个类似的假定。问需要什么共行程序链接？（同(4)作比较。）

▶ 6. [*22*] 对于三个共行程序 A，B 和 C 的情况，给出类似于(1)的共行程序链接，其中的每一个可以转移到其它两个中的任一个。（每当一个共行程序被激活时，它从它上次离开处开始。）

▶ 7. [*30*] 编写一个 MIX 程序，它颠倒正文中的程序所完成的翻译；即你的程序应当把像(3)那样穿了孔的卡片转换成像(2)那样穿了孔的卡片。输出应当是尽可能短的一个字符串，使得(2)中 Z 之前的零将不真正地由(3)产生出来。

1.4.3 解释性程序

在这一小节里，我们将研究一种普通类型的计算机程序——*解释性程序*（它将被简称为*解释程序*）。解释性程序是执行另一个程序的指令的计算机程序，其中的另一个程序是以某种类机器语言写成的。所谓的类机器语言，指的是表示指令的某种方式，其中的指令典型地有操作码、地址，等等。（这个定义和当今大多数计算机术语的定义一样，是不精确的，当然，也没必要那么精确；我们根本不可能精确地划出分界线说哪些程序是解释程序，而哪些不是。）

从历史上说，最初的解释程序是围绕着专门为简单的程序设计而设计的类机器语言而构造的；这样的语言比起真的机器语言更易于使用。用于程序设计的符号语言出现不久就消除了对于这类解释性程序的需要，但是这绝不意味着解释程序开始绝迹了。相反，解释性程序仍在继续发展，它们可以被认为是现代程序设计的必不可少的特征之一。解释程序新的应用，主要是由下列原因引起的：

a) 一种类机器语言有能力以紧凑有效率的方式表示一个复杂的决策和动作序列。

b) 这样一个表示提供了在多遍扫描过程的诸遍之间通信的卓越方式。

在这样一些情况下，专用的类机器语言被建立供在特定的程序中使用。而在那些语言下的程序通常仅由计算机生成。（今天的程序员行家也是好的机器设计者，因为他们不仅建立一个解释性程序，而且他们也定义了其语言有待解释的*虚拟机器*。）

解释性技术有相对地独立于机器的进一步的优点——当改动计算机时，只有解释

程序必须改写。而且,有用的调试辅助手段可以很容易地被构造于一个解释性系统之中。

在这套书中的好多地方都有类型 a)的解释程序的例子出现。例如,参见第 8 章中的递归解释程序和第 10 章中的"语法分析机器"。我们通常需要处理有大量特殊情形但全都类似,然而却没有真正简单的模式这样一种情况。

例如,考虑编写一个代数编译程序,其中我们要生成把两个数量加在一起的有效的机器语言指令。可能有十类的量(常量、简单变量、临时存储单元、下标变量、累加器或变址寄存器的内容、定点或浮点等等)。因此所有各对的组合生成 100 种不同的情况。这就需要一个长的程序以便在每种情况下都来做适当的事情。这个问题的解释性答案是组成其"指令"可装入一个字节的特定语言,然后我们只须编制在这个语言之下的"100 个程序"的表,其中每个程序都理想地装入一个字之中。随后的办法是挑出适当的表项并执行在那里被找到的程序。这一技术简单而有效率。

类型 b)的解释程序例子出现于 D. E. Knuth 的文章"Computer-Drawn Flowcharts", *CACM* **6** (1963),555 ~ 563。在一个多遍程序中,较早的遍必须向稍后的遍传输信息。这个信息作为稍后的遍的一组指令,通常在类机器语言中可最有效地传输;稍后的遍便只不过是专用的解释性程序,而较早的遍是一个专用的"编译程序"。多遍操作的这个原理可以被表征为每当可能,就告知稍后的遍做什么,而不是简单地向它提供大量的事实并且要求它想出要做什么。

类型 b)的解释程序的另一个例子在同特殊语言的编译程序相关联中出现。如果这种语言包含许多特征,这些特征在机器上是不易实现的,除非通过子程序,则得到的目标程序将是非常长的子程序调用序列。例如,如果语言主要是涉及多精度的算术运算,就将发生这种情况。而在这样的情况下,如果以一种解释性语言来表达它,则目标程序将会短得多。例如,参见 B. Randell 和 L. J. Russell 所著的*ALGOL 60 Implementation* (New York:Academic Press,1964),书中描述了把 ALGOL 60 翻译成一种解释性语言的编译程序,它也描述了该语言的解释程序;关于在一个编译程序内部使用的解释程序的例子,还可参见 Arthur Evans Jr.的"An ALGOL 60 Compiler",*Ann. Rev. Auto. Programming* **4** (1964),87 ~ 124。微程序设计机器和专用集成电路芯片的出现已经使得这种解释性方法甚至更有价值。

$\TeX$ 程序(正是它产生了你现在正在阅读的本书的页面*)把包含这一小节的正文的文件换成称为 DVI 格式的解释性语言。这种语言是由 D. R. Fuchs 于 1979 年设计的。[参见 D. E. Knuth,*$\TeX$: The Program* (Reading, Mass.: Addison-Wesley, 1986), Part 31。] $\TeX$ 产生的 DVI 文件然后由称为 dvips 的一个解释程序所处理。该解释程序是由 T. R. Rokicki 编制的,而所产生的文件又在称为 PostScript* 的另一个解释程序中转换成指令文件[Adobe System Inc.,*PostScript Language Reference Manual*, 第 2 版 (Reading, Mass.:Addison-Wesley,1990)]。PostScript 文件被寄给出版商,出版商把它交给一个商业

---

* 指本书的英文版。——译者注

印刷商。他使用 PostScript 解释程序产生印版。这三遍的操作说明了类型 b)的解释程序；T_EX 本身也包含一个小的类型 a)的解释程序，以处理每种类型的字型下字符的所谓连字和出格字信息[*T_EX: The Program*, §545]。

还有另外一种考察以解释性语言写成的程序的方式：它可以当成一个接一个的子程序调用序列。这样一个程序实际上可以被扩展成对子程序调用的一个长的序列，而且，反过来，这样一个序列通常可以被组装成易于解释的编码形式。解释性技术的优点是表示的紧凑性、机器无关性，以及增强的诊断能力。一个解释程序通常可以写成：使得花在代码本身的解释以及转移到适当的程序的时间可以忽略不计。

1.4.3.1 一个 MIX 模拟程序 当提供给一个解释性程序的语言是另一台计算机的机器语言时，则该解释程序通常称为一个*模拟程序*(有时也叫*仿真程序*)。

依作者的意见，程序员花费在编写这样的模拟程序的时间，以及计算机在使用它们中所浪费的时间，实在是太多了。搞模拟程序的动机是简单的：一个计算机中心购置了新的机器，但仍想要运行为旧机器所编写的程序(而不是重写这些程序)。然而比起临时雇用专门的程序员队伍重新进行程序设计来，这通常费用更高而且所得到的结果却更差。例如，作者曾经参与过这样一个重新进行程序设计的项目，结果发现了原来程序中的一个严重错误，而这个程序已经被使用好多年了；新的程序除了换成正确的答案之外，其运行速度是老的程序速度的五倍！(并不是所有的模拟程序都是坏的；例如，在一台新的机器被制造出来之前，计算机厂家先对它进行模拟通常是有利的，使得新机器的软件可以尽可能快地编制出来。但这是非常特殊的应用。)计算机模拟程序的低效率使用有一个极端的例子，实际上真有这样的故事：机器 A 模拟机器 B，而机器 B 则运行一个模拟机器 C 的程序！这就导致了一台大型的昂贵计算机，却给出比其更便宜的同类计算机更差的结果。

鉴于所有这些，那为什么还让这样一个模拟程序在本书中暴露它丑陋的面孔呢？有两个原因：

a)下面将要描述的模拟程序是典型的解释性程序的一个好例子；这里说明了在解释程序中使用的基本技术。它也说明在一个适度长的程序中子程序的用法。

b)我们还将描述以 MIX 语言(用到它所有的东西)写成的 MIX 计算机的模拟程序。这将便于对类似于 MIX 的大多数计算机来编写 MIX 模拟程序；我们的程序的编码有意地避免大量使用面向 MIX 的特征。一个 MIX 模拟程序作为同本书甚至于可能还有其它书相关联的教学辅助工具将是有利的。

本小节中描述的计算机模拟程序应当区别于*离散系统模拟程序*。离散系统模拟程序是重要的程序，我们将在 2.2.5 小节中讨论它们。

现在我们转到编写一个 MIX 模拟程序的任务上。程序的输入将是存储于单元 0000～3499 的一个 MIX 指令和数据序列。我们要模拟 MIX 硬件的精确行为，就当是 MIX 本身正在解释那些指令；因此，我们要实现在 1.3.1 小节中所写的那些特征描述。例如，我们的程序将维持称做 AREG 的一个变量，它将保持被模拟的 A 寄存器的数量；另一个变量 SIGNA 将保存相应的符号。称做 CLOCK 的一个变量将记录在被模拟程序

执行期间,已经经历了多少单位的 MIX 的模拟时间。

MIX 的指令 LDA,LD1,…,LDX 及其它指令的编号提示,我们在连续的单元保持这些寄存器的模拟内容如下:

AREG,I1REG,I2REG,I3REG,I4REG,I5REG,I6REG,XREG,JREG,ZERO

这里 ZERO 是一个在所有时间里都装着零的“寄存器”。JREG 和 ZERO 的位置通过指令 STJ 和 STZ 的操作码号来提示。

在保持我们写模拟程序的哲学——就像它实际上不是由 MIX 的硬件所完成那样——这一过程中,我们将把符号看做一个寄存器的独立部分。例如,许多计算机都不能表示“负零”这个数,而 MIX 肯定能;因此我们在这个程序中将总是特殊地处理符号。单元 AREG,I1REG,…,ZERO 将总是包含相应的寄存器内容的绝对值;在我们的程序中另外的单元组,称 SIGNA,SIGN1,…,SIGNZ,将依赖于相应寄存器的符号是正还是负而包含 +1 或 -1。

一个解释性程序一般地都有一个中央控制部分,它在被解释的指令之间工作。在我们的情况下,在每一个被模拟指令的末尾,程序转移到单元 CYCLE。

控制程序做所有指令都要做的那些事情,把指令拆成不同部分,并把这些部分放到方便的位置以供以后使用。以下的程序设置

rI6 = 下一条指令的位置;

rI5 = M(当前指令的地址,加上变址);

rI4 = 当前指令的操作码;

rI3 = 当前指令的 F 字段;

INST = 当前指令。

程序 M

```
001  *      MIX   SIMULATOR
002               ORIG    3500          被模拟的内存单元从 0000 开始
003  BEGIN        STZ     TIME(0:2)
004               STZ     OVTOG         OVTOG 是被模拟的溢出开关
005               STZ     COMPI         COMPI,±1,或 0,是比较指示符
006               ENT6    0             从单元 0 取头一条指令
007  CYCLE        LDA     CLOCK         控制程序开始
008  TIME         INCA    0             这个地址被置成前一条指令的执行
009               STA     CLOCK           时间(见 033 行)
010               LDA     0,6           rA←欲模拟的指令
011               STA     INST
012               INC6    1             地址计数器加 1
013               LDX     INST(1:2)     取地址的绝对值
014               SLAX    5             把符号附加到地址上
015               STA     M
```

```
016        LD2    INST(3:3)          检查变址字段
017        J2Z    1F                 它为零吗?
018        DEC2   6
019        J2P    INDEXERROR         指定了非法的变址吗?
020        LDA    SIGN6,2            取变址寄存器的符号
021        LDX    16REG,2            取变址寄存器的数值
022        SLAX   5                  附加上符号
023        ADD    M                  对变址做带符号的加法
024        CMPA   ZERO(1:3)          结果是否太大?
025        JNE    ADDRERROR          若是,则模拟一个错误
026        STA    M                  否则,已找到地址
027  1H    LD3    INST(4:4)          rI3←F 字段
028        LD5    M                  rI5←M
029        LD4    INST(5:5)          rI4←C 字段
030        DEC4   63
031        J4P    OPERROR            操作码≥64 码?
032        LDA    OPTABLE,4(4:4)     从表中取出执行时间
033        STA    TIME(0:2)
034        LD2    OPTABLE,4(0:2)     取适当程序的地址
035        JNOV   0,2                转到操作符
036        JMP    0,2                (防止溢出)  ▌
```

请特别注意行 034~036：64 个操作符的一个“切换表”是模拟程序的一部分,允许模拟程序很容易地转到对于当前指令的正确程序处。这是一个重要的节省时间的技术(参见习题 1.3.2-9)。

称做 OPTABLE 的 64 个字的切换表,还给出对于各个操作符的执行时间;以下诸行指出该表的内容:

```
037        NOP    CYCLE(1)           操作码表
038        ADD    ADD(2)               典型的表项格式是
039        SUB    SUB(2)               “OP 程序(时间)”
040        MUL    MUL(10)
041        DIV    DIV(12)
042        HLT    SPEC(1)
043        SLA    SHIFT(2)
044        MOVE   MOVE(1)
045        LDA    LOAD(2)
046        LD1    LOAD,1(2)
           ...
```

```
051          LD6     LOAD,1(2)
052          LDX     LOAD(2)
053          LDAN    LOADN(2)
054          LD1N    LOADN,1(2)
             ...
060          LDXN    LOADN(2)
061          STA     STORE(2)
             ...
069          STJ     STORE(2)
070          STZ     STORE(2)
071          JBUS    JBUS(1)
072          IOC     IOC(1)
073          IN      IN(1)
074          OUT     OUT(1)
075          JRED    JRED(1)
076          JMP     JUMP(1)
077          JAP     REGJUMP(1)
             ...
084          JXP     REGJUMP(1)
085          INCA    ADDROP(1)
086          INC1    ADDROP,1(1)
             ...
092          INCX    ADDROP(1)
093          CMPA    COMPARE(2)
             ...
100 OPTABLE  CMPX    COMPARE(2)      ▮
```

（对于操作符 LD*i*，LD*i*N 以及 INC*i* 的表项有一个额外的“,1”，用于把字段(3:3)设置成非零；这在下面的行 289～290 中使用以指出这样一个事实，即在相应的变址寄存器中的量的大小在模拟这些操作之后必须加以检查。）

我们的模拟程序的下一部分仅仅列出用来存放被模拟的寄存器内容的单元：

```
101          AREG    CON 0        A 寄存器的数值
102          I1REG   CON 0        变址寄存器的数值
             ...
107          I6REG   CON 0
108          XREG    CON 0        X 寄存器的数值
109          JREG    CON 0        J 寄存器的数值
110          ZERO    CON 0        常数 0，供“STZ”使用
```

```
        SIGNA   CON 1          A 寄存器的符号
        SIGN1   CON 1          变址寄存器的符号
        ...
        SIGN6   CON 1
        SIGNX   CON 1          X 寄存器的符号
        SIGNJ   CON 1          J 寄存器的符号
        SIGNZ   CON 1          由“STZ”所存的符号
        INST    CON 0          被模拟的指令
        COMPI   CON 0          比较指示器
        OVTOG   CON 0          溢出开关
        CLOCK   CON 0          模拟的执行时间 ▌
```

现在我们将考虑被模拟程序使用的三个子程序。头一个是 MEMORY 子程序：

调用序列：JMP　MEMORY。

入口条件：rI5 = 正确的内存地址（否则此子程序将转到 MEMERROR）。

出口条件：rX = 内存单元 rI5 中的字的符号；rA = 内存单元 rI5 中的字的数值。

```
 * SUBROUTINES
MEMORY  STJ     9F             内存取子程序
        J5N     MEMERROR
        CMP5    =BEGIN=        被模拟的内存在单元 0000 ~
        JGE     MEMERROR         BEGIN - 1 中
        LDX     0,5
        ENTA    1
        SRAX    5              rX←字的符号
        LDA     0,5(1:5)       rA←字的数值
9H      JMP     *              出口 ▌
```

FCHECK 子程序处理一部分字段的描述，确保它有 8L + R 的形式且 L≤R≤5。

调用序列：JMP FCHECK。

入口条件：rI3 = 正确的字段描述（否则这子程序将转到 FERROR）。

出口条件：rA = rI1 = L，rX = R。

```
FCHECK  STJ     9H             字段检查子程序：
        ENTA    0
        ENTX    0,3            rAX←字段描述
        DIV     =8=            rA←L,X←R
        CMPX    =5=            R > 5 吗？
        JG      FERROR
        STX     R
        STA     L
```

```
143              LD1     L                 rI1←L
144              CMPA    R
145   9H         JLE     *                 出口,除非 L>R
146              JMP     FERROR    ▌
```

最后一个子程序 GETV,求用于各种 MIX 操作符的量 V(即单元 M 的适当字段),如同在 1.3.1 小节所定义的那样。

调用序列:JMP GETV。

入口条件:rI5 = 正确内存地址; rI3 = 正确字段。(如果不正确,将像上面那样,检测出一个错误来。)

出口条件:rA = V 的数值; rX = V 的符号; rI1 = L; rI2 = − R。

第二个入口:JMP GETAV,仅用于比较操作符来从一个寄存器抽取一个字段。

```
147   GETAV      STJ     9F                特殊入口,见行 300
148              JMP     1F
149   GETV       STJ     9F                找 V 的子程序
150              JMP     FCHECK            处理字段并置 rI1←L
151              JMP     MEMORY            rA←内存数值,rX←符号
152   1H         J1Z     2F                符号被包括在此字段中吗?
153              ENTX    1                 若不,则置符号为正
154              SLA     -1,1              把此字段左边所有字节清零
155              SRA     -1,1
156   2H         LD2N    R                 右移到适当位置
157              SRA     5,2
158   9H         JMP     *                 出口  ▌
```

现在我们进入处理每个具体操作符的程序。在这里给出这些程序是为了完整性,但读者只须读其中的一些就行了,除非由于特别感兴趣才仔细地考察它们;建议把 SUB 和 JUMP 操作符作为典型例子来加以研究。注意把类似的操作的程序巧妙地组合在一起的方式,并注意 JUMP 程序如何使用另一个开关表来管理转移的类型。

```
159   *   INDIVIDUAL   OPERATORS
160   ADD        JMP     GETV              取 rA 和 rX 中 V 的值
161              ENT1    0                 令 rI1 指 A 寄存器
162              JMP     INC               转到"增值"程序
163   SUB        JMP     GETV              取 rA 和 rX 中 V 的值
164              ENT1    0                 令 rI1 指 A 寄存器
165              JMP     DEC               转到"减值"程序
166   *
167   MUL        JMP     GETV              取 rA 和 rX 中 V 的值
168              CMPX    SIGNA             符号相同吗?
```

```
         ENTX    1
         JE      *+2           把 rX 置为结果的符号
         ENNX    1
         STX     SIGNA         把它放进两个被模拟的寄存器中
         STX     SIGNX
         MUL     AREG          操作数相乘
         JMP     STOREAX       存入数值
*
DIV      LDA     SIGNA         置余数的符号
         STA     SIGNX
         JMP     GETV          取 rA 和 rX 中 V 的值
         CMPX    SIGNA         符号相同吗?
         ENTX    1
         JE      * +2          把 rX 置为结果的符号
         ENNX    1
         STX     SIGNA         把它放入模拟的 rA 中
         STA     TEMP
         LDA     AREG          操作数相除
         LDX     XREG
         DIV     TEMP
STOREAX  STA     AREG          存数值
         STX     XREG
OVCHECK  JNOV    CYCLE         溢出发生了吗?
         ENTX    1             如果是,则把被模拟的溢出开关置位
         STX     OVTOG
         JMP     CYCLE         返回控制程序
*
LOADN    JMP     GETV          取 rA 和 rX 中 V 的值
         ENT1    47,4          rI1←C-16,表示寄存器
LOADN1   STX     TEMP          使符号变负
         LDXN    TEMP
         JMP     LOAD1         把 LOADN 变成 LOAD
LOAD     JMP     GETV          取 rA 和 rX 中 V 的值
         ENT1    55,4          rI1←C-8,表示寄存器
LOAD1    STA     AREG,1        存数值
         STX     SIGNA,1       存符号
         JMP     SIZECHK       检查是否数值太大
*
```

```
STORE   JMP    FCHECK        rI1←L
        JMP    MEMORY        取存储单元的内容
        J1P    1F            符号是否被包括在字段中?
        ENT1   1             若是,则把 L 改成 1 并"存"寄存器的
        LDX    SIGNA+39,4      符号
1H      LD2N   R             rI2←-R
        SRAX   5,2           把该区域存到字段的右边
        LDA    AREG+39,4     把寄存器插入该字段
        SLAX   5,2
        ENN2   0,1           rI2←-L
        SRAX   6,2
        LDA    0,5           把此区域恢复到字段左边
        SRA    6,2
        SRAX   -1,1          附加上符号
        STX    0,5           存入存储区中
        JMP    CYCLE         返回控制程序
*
JUMP    DEC3   9             转移操作符
        J3P    FERROR        F 是否太大?
        LDA    COMPI         rA←比较指示器
        JMP    JTABLE,3      转到适当的程序
JMP     ST6    JREG          置被模拟的 J 寄存器
        JMP    JSJ
        JMP    JOV
        JMP    JNOV
        JMP    LS
        JMP    EQ
        JMP    GR
        JMP    GE
        JMP    NE
JTABLE  JMP    LE            转移表结束
JOV     LDX    OVTOG         检查是否溢出时
        JMP    *+3             转移
JNOV    LDX    OVTOG
        DECX   1             取溢出开关的补值
        STZ    OVTOG         关闭溢出开关
        JXNZ   JMP           转移
        JMP    CYCLE         不转移
```

```
LE       JAZ    JMP           若 rA 为零或为负则转移
LS       JAN    JMP           若 rA 为负转移
         JMP    CYCLE         不转移
NE       JAN    JMP           若 rA 为负或为正转移
GR       JAP    JMP           若 rA 为正转移
         JMP    CYCLE         不转移
GE       JAP    JMP           若 rA 为正或为零转移
EQ       JAZ    JMP           若 rA 为零转移
         JMP    CYCLE         不转移
JSJ      JMP    MEMORY        检查有效存储地址
         ENT6   0,5           模拟一个转移
         JMP    CYCLE         返回主控程序
*
REGJUMP  LDA    AREG+23,4     寄存器转移:
         JAZ    *+2           寄存器为零吗?
         LDA    SIGNA+23,4    若不,则把符号放进 rA 中
         DEC3   5
         J3NP   JTABLE,3      改变成带条件的 JMP,除非
         JMP    FERROR          F 描述太大
*
ADDROP   DEC3   3             地址转移操作符
         J3P    FERROR        F 是否太大?
         ENTX   0,5
         JXNZ   *+2           找 M 的符号
         LDX    INST
         ENTA   1
         SRAX   5             rX←M 的符号
         LDA    M(1:5)        rA←M 的数值
         ENT1   15,4          rI1 表示寄存器
         JMP    1F,3          四路转移
         JMP    INC           增值
         JMP    DEC           减值
         JMP    LOAD1         入口
1H       JMP    LOADN1        负的入口
DEC      STX    TEMP          符号取反
         LDXN   TEMP          把 DEC 变为 INC
INC      CMPX   SIGNA,1       加法程序
         JE     1F            符号相同吗?
```

```
            SUB    AREG,1          不;数值相减
            JANP   2F              需要改变符号吗?
            STX    SIGNA,1         改变寄存器的符号
            JMP    2F
1H          ADD    AREG,1          数值相加
2H          STA    AREG,1(1:5)     存结果的数值
SIZECHK     LD1    OPTABLE,4(3:3)  我们刚刚已装入一个变址寄存器吗?
            J1Z    OVCHECK
            CMPA   ZERO(1:3)       如果是,确保结果可装入两个字节中
            JE     CYCLE
            JMP    SIZEERROR
*
COMPARE     JMP    GETV            取 rA 和 rX 中的 V 值
            SRAX   5               附加符号
            STX    V
            LDA    XREG,4          取适当寄存器的 F 字段
            LDX    SIGNX,4
            JMP    GETAV
            SRAX   5               附加符号
            CMPX   V               比较(注意 -0 = +0)
            STZ    COMPI           把比较指示器置为零,
            JE     CYCLE             正 1,或负 1
            ENTA   1
            JG     *+2
            ENNA   1
            STA    COMPI
            JMP    CYCLE           返回控制程序
*
            END    BEGIN   ▌
```

上边的代码坚持了在 1.3.1 小节指出的一条微妙的规则:即当变址寄存器 1 包含 +5 时,指令“ENTA -0”就像“ENTA -5,1”所做的那样,把负零装入到寄存器 A 中。一般地说,当 M 为零时,ENTA 装入指令的符号,而 ENNA 装入相反的符号。当作者编写 1.3.1 小节的初稿时,曾经忽略了说明这一条件的必要性;通常仅当要编写出的计算机程序须遵循这些规则时这样的问题才冒出来。

尽管这个程序已经很长了,但是在若干方面它还是不完备的:

a)它不能识别浮点运算。

b)关于操作码 5,6 和 7 的编码被留作习题。

c)关于输入输出操作符的编码被留作习题。

d)未能提供被模拟程序的装入(见习题 4)。

e)未包括出错程序

INDEXERROR,ADDRERROR,OPERROR,MEMERROR,FERROR,SIZEERROR

这些程序处理在被模拟的程序中所探测到的错误条件。

f)没有提供诊断措施。(例如,一个有用的模拟程序应当使得有可能在程序正在执行时打印出寄存器的内容。)

习 题

1. [*14*]试研究在模拟程序中 FCHECK 子程序的所有用法。你能否给出更好地组织这个代码的建议?(请见在 1.4.1 小节末尾的讨论中的步骤 3。)

2. [*20*]试写出 SHIFT 程序,它在正文的程序中未出现(操作码 6)。

▶ 3. [22] 试写出 MOVE 程序,它在正文的程序中未出现(操作码 7)。

4. [*14*] 试改动正文中的程序使得它如同 MIX 的"GO 按钮"已被按下那样来开始(见习题 1.3.1-26)。

▶ 5. [*24*] 同 MIX 本身直接执行 LDA 和 ENTA 操作符所花费的实际时间作比较,试确定模拟这些操作符所需要的时间。

6. [*28*] 写出在正文中的程序里未给出的针对输入输出操作符 JBUS,IOC,IN,OUT 和 JRED 的程序,但只允许使用 16 和 18 两个设备。假设操作"读卡"和"跳到新页"耗时 $T=10000u$。而"打印行"耗时 $T=7500u$。(注:经验表明,JBUS 应把"JBUS *"当做一个特殊情况来加以模拟;否则模拟程序似乎要停止!)

▶ 7. [*32*] 修改上一题的解:使 IN 或 OUT 的执行不引起立即的输入输出传输;传输应当在被模拟设备所要求的时间过了大约一半之后才开始进行。(这将防止学生不适当地使用 IN 和 OUT 的操作码而经常出现错误。)

8. [*20*] 是真还是假:每当执行模拟程序的 010 行时,我们就有 $0\leq rI6<$ BEGIN。

***1.4.3.2 跟踪程序** 当在一台机器上模拟它自己时(如同在上一小节中在 MIX 上模拟 MIX 那样),就出现了模拟程序的一种特殊情况,这种模拟程序称做*跟踪*或*监控*程序。这样的程序有时被用来帮助调试,因为它们打印出被模拟的程序如何动作的一步一步的记述。

上一小节的程序已被写成就像另一个计算机模拟 MIX 那样。对跟踪程序使用的是十分不同的方法;我们一般地让寄存器表示它们自己,并且让操作符也实现它们自己。事实上,我们通常装做让机器由其本身执行大多数的指令。主要的例外是转移或条件转移指令,如果不修改它们就不能被执行,因为跟踪程序必须保持处于控制之中。每台机器也都有使跟踪更具挑战性的特性;在 MIX 的情况下,J 寄存器就成了最有趣的问题。

当主程序转移到单元 ENTER,且寄存器 J 置成*启动*跟踪的地址,寄存器 X 置成跟踪应当停止的地址时,就启动以下给出的跟踪程序。这个程序是有趣的,值得仔细研究。

```
*   TRACE   ROUTINE
ENTER    STX    TEST(0:2)         置出口单元
         STX    LEAVEX(0:2)
         STA    AREG              保存 rA 的内容
         STJ    JREG              保存 rJ 的内容
         LDA    JREG(0:2)         取供跟踪的起始地址
CYCLE    STA    PREG(0:2)         保存下一指令的地址
TEST     DECA   *                 是出口地址吗?
         JAZ    LEAVE
PREG     LDA    *                 取下一指令
         STA    INST              复制它
         SRA    2
         STA    INST1(0:3)        保存地址和变址部分
         LDA    INST(5:5)         取操作码 C
         DECA   38
         JANN   1F                C≥38(JRED)吗?
         INCA   6
         JANZ   2F                C≠32(STJ)吗?
         LDA    INST(0:4)
         STA    *+2(0:4)          把 STJ 改成 STA
JREG     ENTA   *                 rA←被模拟的 rJ 的内容
         STA    *
         JMP    INCP
2H       DECA   2
         JANZ   2F                C≠34(JBUS)吗?
         JMP    3F
1H       DECA   9                 测试转移指令
         JAP    2F                C>47(JXNP)吗?
3H       LDA    8F(0:3)           我们检测到一个转移指令;
         STA    INST(0:3)           把它的地址改成"JUMP"
2H       LDA    AREG              恢复寄存器 A
*                                 所有寄存器,但 J 除外,相对于外部
*                                   程序现在都有适当的值
INST     NOP    *                 指令被执行
         STA    AREG              再次存寄存器 A
INCP     LDA    PREG(0:2)         移到下一指令
         INCA   1
         JMP    CYCLE
```

```
39  8H      JSJ    JUMP          用于行 29 和行 40 的常数
40  JUMP    LDA    8B(4:5)       已出现一个转移
41          SUB    INST(4:5)     是 JSJ 吗?
42          JAZ    * + 4
43          LDA    PREG(0:2)     若不然,修改被模拟的 J 寄存器
44          INCA   1
45          STA    JREG(0:2)
46  INST1   ENTA   *
47          JMP    CYCLE         移到转移的地址
48  LEAVE   LDA    AREG          恢复寄存器 A
49  LEAVEX  JMP    *             停止跟踪
50  AREG    CON    0             被模拟的 rA 的内容   ▌
```

关于一般的跟踪程序以及特别是这一个跟踪程序,有下列事情应该说明:

1)我们还仅仅介绍了一个跟踪程序最有趣的部分,即当执行另一个程序时还保留控制的部分。要使一个跟踪程序有用,必须有写出诸寄存器内容的一个程序,而这一点还未被包括进来。这样一个程序会分散对于一个跟踪程序更微妙的特性的注意,尽管它肯定也很重要;必要的修改留作习题(见习题 2)。

2)空间一般比时间更重要,即应把程序写得尽可能地短。这样跟踪程序将有可能同极端长的程序共存。无论如何运行时间是由输出消耗掉的。

3)为避免破坏大多数寄存器的内容,应小心从事;事实上,这个程序只使用 MIX 的 A 寄存器。跟踪程序不会影响比较指示器也不会影响溢出开关。(寄存器使用越少,我们需要做的恢复也越少。)

4)当出现对单元 JUMP 的转移时,不需要"STA AREG",因为 rA 未曾改变。

5)在离开了跟踪程序之后,J 寄存器未被适当地还原,习题 1 说明如何修改这一点。

6)被跟踪的程序仅受三个限制的支配:

a)它必须不把任何东西存到由跟踪程序所使用的单元。

b)它必须不使用记录着跟踪信息的输出设备(例如,JBUS 将给出一个不适当的指示)。

c)当被跟踪时,它将以较低的速度运行。

习　题

1. [*22*] 试修改正文中的跟踪程序,使得当离开时它恢复寄存器 J。(可以假定寄存器 J 不是零。)

2. [*26*] 试修改正文中的跟踪程序,使得在执行每个程序步骤之前它在磁带单元 0 上写以下的信息:

字 1,(0:2)字段:单元。

字 1,(4:5)字段:寄存器 J(执行之前)。

字 1,(3:3)字段:如果比较结果是大于则为 2,如果是等于则为 1,如果是小于则为 0;如果执行之前溢出开关不置位则加 8。

字 2:指令。

字 3:寄存器 A (执行之前)。

字 4~9:寄存器 I1~I6 (执行之前)。

字 10:寄存器 X (执行之前)。

每 100 个字的磁带的块中的字 11~100 应该有相同的格式,它们包含 9 个 10 个字的组。

3.[*10*] 上一道题提出,跟踪程序把它的输出写到磁带上。试讨论为什么这比直接打印更可取。

▶ 4.[*25*] 如果跟踪程序跟踪它自己那将发生什么情况?特别地,如果把两个指令 ENTX LEAVEX;JMP *+1 放在 ENTER 的前面,试考虑其行为。

5.[*28*] 以类似于用来求解上一道题的方式,试考虑把跟踪程序的两个副本放在内存中的不同位置,而且把每个设置成去跟踪另一个。那将发生什么情况?

▶ 6.[*40*] 试写出在习题 4 的意义下,一个有能力来跟踪自己的跟踪程序:它应当以较慢的速度打印出它自己程序的步骤,而且该程序将仍然以更慢的速度跟踪它自己,无穷地做下去,直到超过内存的容量为止。

▶ 7.[*25*] 试讨论怎样写一个有效的转移跟踪程序,它比一个通常的跟踪产生少得多的输出。一个转移跟踪不显示寄存器的内容,而是简单地记录出现的转移,它输出数偶序列 (x_1,y_1),(x_2,y_2),…,意思是程序是从单元 x_1 转移到 y_1,然后(在实现了在单元 y_1, y_1+1,…,x_2 处的指令之后),从单元 x_2 转移到 y_2,等等。[根据这一信息,随后的程序有可能重新构造程序的流程,并推断每个指令是如何频繁地被执行的。]

1.4.4　输入和输出

在一台计算机和另一台计算机之间最突出的区别或许是可用的输入和输出设备,以及支配这些外围设备的计算机指令。我们不可能期望在一本书里全部讨论这一领域中的所有问题和技术,因此我们将限于研究用于大多数计算机的典型的输入输出方法。MIX 的输入输出操作符代表了在实际的机器中可利用的广泛变化的设备之间的折中;为帮助理解输入输出,且让我们在这一小节里讨论获得最好的 MIX 输入输出的问题。

作者再次向读者请求对于成了老黄历的 MIX 计算机以及它的穿孔卡片等等的宽容。尽管这种老式的设备现在已非常陈旧了,但它们仍可用于讲授重要的内容。当然当 MMIX 计算机出现时,用于讲授这些内容会更好些。

许多计算机用户都觉得输入和输出实际上不是"实际的"程序设计的一部分;输入和输出被认为是繁琐的任务,仅仅因为人们需要使信息进入机器和从机器中读出才必须实现它们。由于这一原因,通常在对于一台计算机的所有其它特征都考察完了之前,都不会去学它的输入和输出设备。因而经常出现的情况是,关于输入和输出的细节,只是一台特定计算机的一小部分程序员才知道得多些。这种局面是自然的,因为机器的输入输出设备从来都不是特别精良。然而,如果没有更多的人对于这一课题予以认真

思考,是不能期望这种状况会有改进的。在这一小节和其它地方(例如,5.4.6 小节)我们将看到,某些非常有趣的事情是同输入输出相关联而出现的,其中还存在着一些令人赏心悦目的算法呢。

在这里先对术语作一点简略的讨论也许是适当的。尽管英文辞典以前只把"input"(输入)和"output"(输出)作为名词列入("What kind of input are we getting?"),但现在已经习惯在文法上把它们用做形容词("Don't drop the input tape")以及作为及物动词("Why did the program output this garbage?")。合并一起的术语"输入输出"最经常地通过缩写的"I/O"来提及。输入通常叫做读,相应地,输出叫做写。输入或输出的素材一般称做"data"(数据),这个词,严格地说,是 datum 的复数形式,但它是集体地使用的,就像它是单数一样("The data has not been read"),这同"information"(信息)一词既是单数也是复数一样。今天的英文课就讲到这里。

现在假设我们希望从磁带输入。如同在 1.3.1 小节所定义的那样,MIX 的 IN 操作符只是启动输入过程。在输入进行的同时计算机继续执行进一步的指令。因此指令"IN 1000(5)"将开始从磁带机第 5 号读入 100 个字到内存单元 1000 ~ 1099,但接下来的程序在以后的一段时间不得访问这些内存单元。只有在(a)访问第 5 号磁带机的另一个 I/O 操作(IN,OUT 或 IOC)已开始,或(b)条件转移指令 JBUS (5)或 JRED (5)指出第 5 号磁带机不再"忙碌"之后,程序才能假定输入完成。

因此,把一个磁带块输入到单元 1000 ~ 1099 中并给出信息的最简单方式是两条指令的序列:

```
IN 1000(5);JBUS *(5)                                          (1)
```

在 1.4.2 小节的程序中我们已经使用过这个基本的方法(见该节行 07 ~ 08 和 52 ~ 53)。然而,这个方式一般是会浪费计算机时间的,因为非常大量潜在地有用的计算时间,比如说 $1000u$ 或甚至 $10000u$,由重复执行"JBUS"指令所消耗。如果这额外的时间用于计算的话,程序的运行速度可能会提高一倍。(见习题 4 和 5。)

对于输入来说避免这样一个"忙碌等候"的方法是使用两个内存区域。我们可以往一个区域读入数据,而用另一个区域内的数据进行计算。例如,可以以指令

```
IN   2000(5)       开始输入第一个块                           (2)
```

开始程序。接着,我们在每当需要一个磁带块时,给出以下五条指令:

```
ENT1   1000        为 MOVE 操作符做准备
JBUS   *(5)        等候直到 5 号设备就绪                       (3)
MOVE   2000(50)    (2000 ~ 2049)→(1000 ~ 1049)
MOVE   2050(50)    (2050 ~ 2099)→(1050 ~ 1099)
IN     2000(5)     开始输入下一个块
```

这和(1)有相同的总体效果,但当程序对单元 1000 ~ 1099 中的数据进行处理时,它保持输入带忙碌。

(3)的最后一条指令在考察前一个块之前开始把一个磁带块输入到单元 2000 ~ 2099 中去。这叫做"提前读",或预先输入——它是在这个块最终将是需要的这样一个信念之下完成的。然而,事实上,在我们开始考察 1000 ~ 1099 的块之后,我们可能发

现,实际上并不需要更多的输入。例如,考虑 1.4.2 小节共行程序中的类似情况,其中输入来自于穿孔卡片而不是带:在卡片上任何地方出现的“.”意味着它是这组卡片的最后一张。这样一种情况将使预先输入不可能进行,除非我们可以假定(a)一张空白卡片或者某个其它种类的特殊的结尾卡片将跟着输入卡片组,或者(b)一个识别标记(例如“.”)将出现在,比如说,这组卡片最后一张的第 80 列处。每当预先进行输入时,总是必须在程序的末尾提供适当地终止输入的某种手段。

使计算时间和 I/O 时间重叠的技术称做缓冲,而基本的方法(1)称做非缓冲输入。用来保存(3)中的预先输入的内存区域 2000 ~ 2099,以及把输入移到其中的区域 1000 ~ 1099,叫做缓冲区(buffer)。《韦伯斯特新世界词典》把“缓冲区”定义为“用于减轻冲击的任何人与物”,因此这个术语是适当的,因为缓冲趋向于使 I/O 设备顺畅地进行。(计算机工程师经常在另外的意义下使用“缓冲区”一词,用它来表示在传输期间存储信息的 I/O 设备的一部分。然而,在本书之中,“缓冲区”将表示程序员保存 I/O 数据所使用的一个内存区域。)

序列(3)并不总比(1)优越,尽管例外情况很少。让我们来比较两者的执行时间:假设 T 是为输入 100 个字所需要的时间,并假设 C 是在输入请求之间插入的计算时间。方法(1)对于每个带块实质上需要 $T+C$ 的时间,而方法(3)实际上花费 $\max(C,T)+202u$ 的时间。(量 $202u$ 是两条 MOVE 指令所需要的时间。)考察这个运行时间的一个方法是考虑“关键路径时间”——在此情况下是 I/O 设备在使用之间的空闲时间数量。方法(1)保持设备空闲的时间是 C 个时间单位,而方法(3)保持设备空闲的时间是 202 个单位(假定 $C<T$)。

(3)的比较缓慢的 MOVE 指令是不合意的,特别是因为它们占据了关键路径时间,使磁带机不得工作。有一个几乎是显然的改进方法,允许我们避免这些 MOVE 指令:可以修改外边的程序使得它交替地访问单元 1000 ~ 1099 和 2000 ~ 2099。当我们向一个缓冲区域输入时,可以对另一个区域中的信息进行计算;然后当对头一个区域中的信息进行计算时,我们可以开始向第二个缓冲区输入。这是称做**缓冲区交换**的重要技术。当前感兴趣的缓冲区的地址将被保存在一个变址寄存器中(或者,如果没有变址寄存器,就保存在一个内存单元中)。我们已经在算法 1.3.2P 中看到用于输出的缓冲区交换的一个例子(见步骤 P9 ~ P11)以及其伴随的程序。

作为用于输入的缓冲区交换的一个例子,假设我们有一计算机应用,其中每个带块由 100 个分开的一个字的项组成。以下程序是一个子程序,它读取输入的下一个字,并在当前块已穷尽时开始在一个新块读。

```
WORDIN  STJ   1F             存储出口地址
        INC6  1              前进到下一个字
2H      LDA   0,6            它是缓冲区的末尾吗?
        CMPA  =SENTINEL=
1H      JNE   *              如不是,出口
        IN    -100,6(U)      重新填满这个缓冲区
        LD6   1,6            转到另一个缓冲区
```

```
08              JMP    2B            并返回                  (4)
09   INBUF1     ORIG   *+100         第一个缓冲区
10              CON    SENTINEL      缓冲区末尾的标记
11              CON    *+1           另一个缓冲区的地址
12   INBUF2     ORIG   * +100        第二个缓冲区
13              CON    SENTINEL      缓冲区末尾的标记
14              CON    INBUF1        另一个缓冲区的地址  ▌
```

在这个程序中,变址寄存器 6 被用于对输入的最后一个字的编址;我们假定,调用程序不影响这个寄存器。符号 U 表示磁带机,而符号 SENTINEL 表示(根据程序的特征)一个所有磁带块中都不会出现的值。

关于这个子程序有好些事要作说明:

1)标记常数作为每个缓冲区的第 101 个字出现,因此它为检查缓冲区的结束提供了方便。然而,在许多应用中,标记技术将不是可靠的,因为任何字都可出现在带上。但如果我们进行卡片输入,却总是可以使用一个类似的方法(以缓冲的第 17 个字等于标记)而不必担心出错;在该情况下,任何负的字可以作为一个标记,因为 MIX 由卡片输入总是给出非负的字。

2)每个缓冲区都包含另一个缓冲区的地址(见行 07,11 和 14)。这种"链接在一起"便于交换过程的进行。

3)不需要有 JBUS 指令,因为在访问上一个块的任何字之前就启动了下一个输入。如果和以前一样用量 C 和 T 表示计算时间和带输入时间,则每个带块的执行时间现在是 $\max(C,T)$;因此有可能使带以全速进行,如果 $C\leq T$ 的话。(注:然而,在这方面 MIX 是一个理想化的计算机,因为程序没有 I/O 错误需要处理。在大多数的计算机中,在这里的"IN"指令之前,都需要测试上一个操作是否成功完成的某些指令。)

4)为使子程序(4)正常工作,当程序开始时,有必要使各种事情正确地开始。其细节留给读者(见习题 6)。

5)WORDIN 子程序使得就这个程序的剩余部分而言,磁带机看起来有长度为 1 的块,而不是长度为 100 的块。将若干个面向程序的记录填入单个实际的带块中的思想称做**记录的分块**。

我们对于输入所说明的技术,只须作小的改动,也适用于输出(见习题 2 和 3)。

多缓冲区 缓冲区交换仅仅是涉及 N 个缓冲区的一般方法的特殊情况($N=2$)。在某些应用中两个以上的缓冲区是合意的;例如,考虑下列类型的算法:

步骤 1. 快速持续读五个块。

步骤 2. 对这些数据执行相当长的计算。

步骤 3. 返回步骤1。

这里五个或六个缓冲区将是合意的,使得在步骤 2 期间可以读入下一批的五个块。这种将 I/O 活动"捆在一起"的办法使得多个缓冲成为对缓冲区交换的一个改进。

假设我们有 N 个缓冲区用于使用单个 I/O 设备的某个输入或输出过程;如图 23 中

所示，我们将想像，缓冲区被安排在一个圈上。缓冲过程外边的程序可被假设对应于感兴趣的 I/O 设备，有下列的一般形式：

```
   ⋮
ASSIGN
   ⋮
RELEASE
   ⋮
ASSIGN
   ⋮
RELEASE
```

换言之，我们可以假定，这个程序在叫做“ASSIGN”（赋值，指定）的动作和叫做“RELEASE”（释放）的动作之间交替进行，两者之间插入不影响缓冲区分配的其它计算。

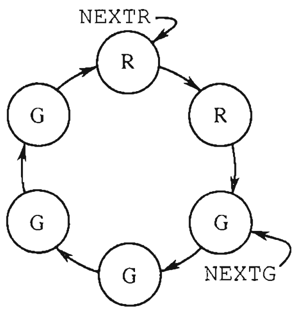


图 23　缓冲区的一个圈（$N=6$）

ASSIGN 表示程序获得下一个缓冲区域的地址；该地址指定为某个程序变量的值。

RELEASE 表示这个程序已完成对当前缓冲区域的处理。在 ASSIGN 和 RELEASE 之间，这个程序和缓冲区之一（称做当前缓冲区域）进行通信；在 RELEASE 和 ASSIGN 之间，这个程序不访问任何缓冲区域。

可以想像，ASSIGN 可以立即跟随 RELEASE，而且关于缓冲区的讨论通常就是基于这个假定的。然而，如果 RELEASE 尽可能快地完成，则缓冲过程会有更多的自由而且也将更有效率；通过把两个实质上不同的函数 ASSIGN 和 RELEASE 分开，我们将发现缓冲技术更易于理解，而且即使当 $N=1$ 时我们的讨论也仍将有意义。

为了更加明确，让我们分开考虑输入和输出的情况。对于输入，假设我们正在处理一台卡片输入机。ASSIGN 动作意味着程序需要看到来自一张新卡片的信息；我们将把一个变址寄存器置成放置下一张卡片映像的内存的地址。当不再需要当前卡片映像中的信息时出现 RELEASE 动作——该信息已由程序消化掉了，或许是被复制到内存的另一部分，等等。因此当前的缓冲区域可装入下一步的预输入数据。

对于输出，考虑一台行式打印机的情况。当需要一个空的缓冲区时，发生 ASSIGN

动作,以便把一行映像放置于该缓冲区中供打印用。我们希望设立一个变址寄存器等于这样一个区域的内存地址。当这个行映像已经在缓冲区域中以可供打印的形式完全建立起来时,出现 RELEASE 动作。

例:为打印 0800~0823 单元的内容,我们可以写

```
JMP  ASSIGNP    (置 rI5 为缓冲区的地址)
ENT1  0,5
MOVE  800(24)  将 24 个字移入输出缓冲区
JMP   RELEASEP
```
(5)

其中 ASSIGNP 和 RELEASEP 表示对行式打印机完成这两个缓冲功能的子程序。

从计算机的观点看,在最优情况下,ASSIGN 操作实际上将不需要执行时间。在输入时,这意味着每个卡片映像都将被预先处理使得当程序已就绪时,数据也已可利用;而在输出时,它意味着在内存中总有空位置来记录行映像。在两种情况下,都不需要等候 I/O 设备。

为了帮助描述缓冲算法,而且也为使它更有声有色,我们将说缓冲区域是绿色、黄色或红色的(在图 24 中标以 G,Y 和 R)。

绿色是指这个区域已为 ASSIGN 做好准备。这表示它已被预先处理的信息所充满(在输入情况下)或者它是一个空区域(在输出情况下)。

黄色是指这个区域已经被 ASSIGN 过了,但还未 RELEASE。这表明它是当前缓冲区,因此程序正同它进行通信。

红色是指这个区域已经被 RELEASE 了。因此它是一个空区域(在输入情况下)或者它已被信息充满了(在输出情况下)。

图 23 示出同缓冲区圈相关联的两个"指针"。从概念上说,这些指针就是程序中的变址寄存器。NEXTG 和 NEXTR 分别指向"下一个绿色"和"下一个红色"缓冲区。第三个指针 CURRENT(示于图 24)表示存在时的黄色缓冲区。

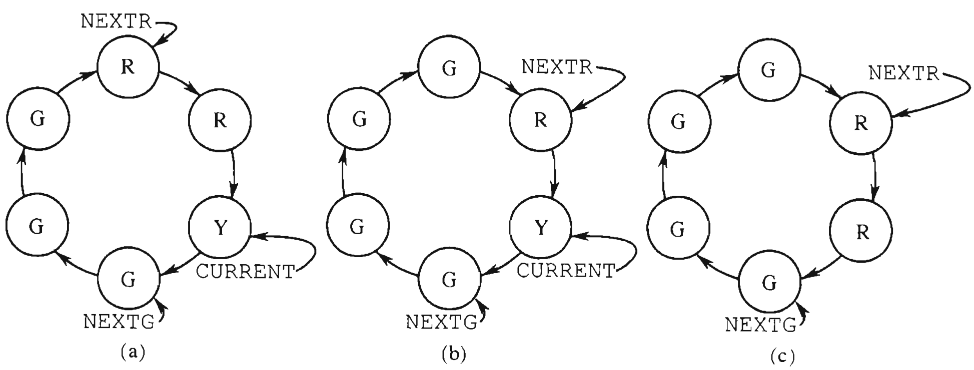


图 24 缓冲区的转换

(a)在 ASSIGN 之后;(b)在 I/O 完成后,(c)在 RELEASE 之后。

以下的诸算法都很适用于输入或输出，但为确定起见，我们将首先考虑来自一台卡片输入机的输入情况。假设一个程序已经达到图 23 所示的状态。这意味着四个卡片映像已由缓冲过程预先处理过了。因此它们驻留在绿色缓冲区中。这时，两件事情同时发生了：(a)跟随着一个 RELEASE 操作之后，程序在进行计算；(b)一张卡片正被读进由 NEXTR 所指示的缓冲区中。这种事态将继续到输入循环完成（该卡片输入设备然后将从“忙碌”转为“就绪”）为止，或者继续到该程序执行一个 ASSIGN 操作为止。假设首先出现的是后一种情况；则由 NEXTG 指示的缓冲区变成黄色（它被指定为当前缓冲区），NEXTG 沿顺时针方向移动，因此我们达到图 24(a)所示的位置。如果现在输入完成，另一个预先处理的块出现；因此缓冲区从红色变成绿色，而且 NEXTR 移动如图 24(b)所示。如果接着而来的是 RELEASE 操作，我们就得到图 24(c)。

对于涉及输出的例子，参见习题中的图 27。这张图说明，在一个从四个快速的输出开始，然后以慢速产生四个输出，而当结束时最后又连续快速地发出两个输出的程序中，缓冲区域的“颜色”是时间的函数。在该例子中出现三个缓冲区。

指针 NEXTR 和 NEXTG 兴高采烈地围绕着圆圈前进，各以独立的速度沿顺时针方向移动。这是程序（它把缓冲区由绿色变成为红色）和 I/O 缓冲过程（它把缓冲区从红色转变成绿色）之间的竞赛。可能出现两种冲突的情况：

a)如果 NEXTG 试图超过 NEXTR，则程序已跑在 I/O 设备的前边，因此它必须等候到该设备就绪。

b)如果 NEXTR 试图超过 NEXTG，则 I/O 设备已经跑在程序前边，因此我们必须关闭它直到给出了下一个 RELEASE 时为止。

图 27 中对这两种情况都做了描述。（参见习题 9。）

幸而，尽管对于缓冲区圆圈背后的思想刚刚给出的说明稍微长些，但是处理这种情况的实际算法却都很简单。在下列的叙述中，

N = 缓冲区总的数目；

n = 红缓冲区的当前数目。

变量 n 在以下的程序中用于避免 NEXTG 和 NEXTR 之间的相互干扰。

算法 A(ASSIGN) 这个算法如同上边所描述的那样，包括在一个计算程序内由 ASSIGN 所蕴涵的步骤。

A1. [等候 $n < N$] 如果 $n = N$，暂停程序直到 $n < N$ 为止。（如果 $n = N$，无缓冲区可供指定；但是以下的算法 B，它同本算法并行运行，将最终成功产生一个绿色缓冲区。）

A2. [CURRENT←NEXTG] 置 CURRENT←NEXTG（由此指定当前缓冲区）。

A3. [推进 NEXTG] 把 NEXTG 推进到顺时针方向的下一个缓冲区。 ▌

算法 R(RELEASE) 如同上面所述，这个算法包括在一个计算程序内由 RELEASE 所蕴涵的步骤。

R1. [n 加 1] n 加 1。 ▌

算法 B(缓冲区控制) 这个算法实现机器中 I/O 操作的实际启动；在下面所述的

意义下,它要同主程序一起“同时”被执行。

B1. [计算] 令主程序计算一小段时间;在延迟某一时间之后,当 I/O 设备已为另一个操作做好准备时,步骤 B2 将被执行。

B2. [$n=0$?] 如果 $n=0$,则转到 B1。(因此,如果没有红色缓冲区,则无 I/O 动作可执行。)

B3. [启动 I/O] 启动由 NEXTR 所指定的缓冲区域和 I/O 设备之间的传输。

B4. [计算] 令主程序运行一段时间;然后当 I/O 操作完成时转到 B5。

B5. [推进 NEXTR] 把 NEXTR 推进到顺时针方向中的下一个缓冲区。

B6. [n 减 1] n 减 1,并转到 B2。 ▌

在这些算法中,我们有两个“同步”地进行的独立的进程,即缓冲控制程序和计算程序。事实上,这些程序是共行程序,我们将把它们叫做 CONTROL 和 COMPUTE。共行程序 CONTROL 在步骤 B1 和 B4 时转到 COMPUTE;共行程序 COMPUTE 通过在程序的零星间隔中穿插“就绪转移”指令而转到 CONTROL。

对 MIX 把这个算法编写成程序是极其简单的。为方便起见,假定把缓冲区链接起来使得在每个缓冲区之前的字是下个缓冲区的地址;例如,对于 $N=3$ 个缓冲区,我们有 CONTENTS(BUF1 - 1) = BUF2,CONTENTS(BUF2 - 1) = BUF3,以及 CONTENTS(BUF3 - 1) = BUF1。

程序 A(ASSIGN,在 COMPUTE 共行程序内的一个子程序) rI4 ≡ CURRENT; rI6 ≡ n; 调用序列是 JMP ASSIGN;在离开时,rX 包含 NEXTG。

```
ASSIGN  STJ   9F           子程序链接
1H      JRED  CONTROL(U)   A1. 等候 n<N
        CMP6  =N=
        JE    1B
        LD4   NEXTG        A2. CURRENT←NEXTG
        LDX   -1,4         A3. 推进 NEXTG
        STX   NEXTG
9H      JMP   *            离开  ▌
```

程序 R(RELEASE,用于 COMPUTE 共行程序内的程序) rI6 ≡ n。每当需要 RELEASE 时就插入这个短的程序。

```
        INC6  1            R1. n加1
        JRED  CONTROL(U)   可能的到 CONTROL 共行程序的转移  ▌
```

程序 B(CONTROL 共行程序) rI6 ≡ n, rI5 ≡ NEXTR。

```
CONT1   JMP   COMPUTE      B1. 计算
1H      J6Z   *-1          B2. n=0?
        IN    0,5(U)       B3. 启动 I/O
        JMP   COMPUTE      B4. 计算
```

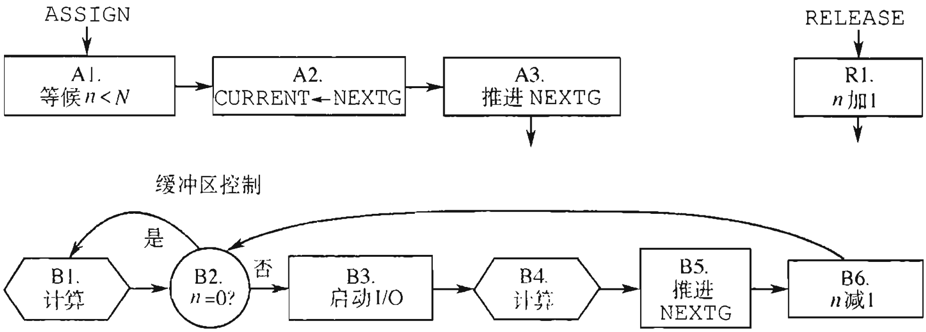


图 25　多缓冲区的算法

```
     LD5     -1,5          B5. 推进NEXTR
     DEC6    1             B6. n减1
     JMP     1B         ▌
```

除上面的程序外,我们还有通常的共行程序链接

```
CONTROL   STJ  COMPUTEX    COMPUTE   STJ  CONTROLX
CONTROLX  JMP  CONT1       COMPUTEX  JMP  COMP1
```

而且指令"JRED CONTROL(U)"在大约每 50 条指令中应被放置于 COMPUTE 中一次。

因此对于多缓冲的程序实际上总共只有 7 条 CONTROL 指令,8 条 ASSIGN 指令以及 2 条 RELEASE 指令。

或许值得注意的是,恰恰是同一个算法对于输入和输出都有效。那区别何在——控制程序如何知道是预先处理(对于输入)还是滞后(对于输出)? 答案在于初始条件:对于输入我们以 $n = N$(所有缓冲区为红色)开始,而对于输出,我们以 $n = 0$(所有缓冲区为绿色)开始。一旦程序已适当地开始,它就分别作为输入过程或输出过程继续动作。另外的初始条件是 NEXTR = NEXTG,两者都指向缓冲区之一。

在程序结束时,有必要停止 I/O 过程(如果它是输入)或者等候到它完成(对于输出);我们把细节留给读者完成(见习题 12 和 13)。

重要的是问,用什么 N 值最好。肯定地说,当 N 变得更大时,程序的速度将不减慢,但它也并不无限地增加,因而我们就会到达开始减慢的点。让我们再次引用数量 C 和 T 来表示 I/O 操作符之间的计算时间和 I/O 时间本身。更确切地说,令 C 是连续两个 ASSIGN 之间的时间,而令 T 是传送一个块所需要的时间。如果 C 总是大于 T,则 $N = 2$ 就足够,因为不难看出,通过两个缓冲区我们就能使计算机总是保持忙碌。如果 C 总小于 T,则 $N = 2$ 仍然足够,因为我们可使输 I/O 设备总保持忙碌(除非像在习题 19 那样,当设备有特别的计时限制时)。因此主要是当 C 是在小的值与大值之间变化时较大的 N 值才有用;如果 C 的大值要比 T 的值持续得久得多,则连续的小值的平均数,加上 1,可能对 N 正合适。(然而,如果所有输入都出现在程序开始时以及所有输出都出现在末尾时,则缓冲的优点就几乎没有了。)如果在 ASSIGN 和 RELEASE 之间的时间

总是十分小,则在整个上面的讨论中,N 的值可以减 1,且它对运行时间影响很小。

这个缓冲方法可以以多种方式加以修改,这里我们将简略地提及其中一些。迄今为止,我们已经假定,被使用的 I/O 设备只有一个;当然,实际上,将同时使用多个设备。

有多种方法来解决多个设备这一课题。在最简单的情况下,我们可以对每个设备都有一个单独的缓冲区圈。每个设备都有它自己的 n, N, NEXTR, NEXTG 和 CURRENT 的值,并且有它自己的 CONTROL 共行程序。这将提供在每个 I/O 设备上同时有效的缓冲动作。

也有可能把相同大小的缓冲区域建成"池",使得两个或多个设备从一个公共的表中共享缓冲区。这可以通过使用第 2 章的链接存储器技术来加以处理,使所有红色的输入缓冲区链在一起成为一个表,而所有绿色的输出缓冲区也链在一起成为另一个表。在这种情况下,就有必要来区分输入和输出了,并且不必使用 n 和 N 来重写这些算法。如果池中所有的缓冲区都被预先读的输入填满,则这个算法就将不可挽回地会梗塞;因此应当做一个检查,使得总是至少有一个缓冲区(最好是每个设备都有一个)不是绿色输入的;仅当 COMPUTE 程序对于某个输入设备在步骤 A1 处停止不前时,才允许从该设备输入到缓冲池的最后一个缓冲区中。

某些机器对于 I/O 设备的使用还有附加的限制,使得从某些设备对不可能同时传送数据。(例如,多个设备可能通过单个"通道"被连接到计算机上)。这个限制也影响我们的缓冲程序;当我们必须选择下一次启动哪一个 I/O 设备时,如何来作选择呢? 这称做"预报"问题。对于一般情况最好的预报规则似乎是把选择的优先权提供给其缓冲区圈有最大的 n/N 值者,并假定已经对圈中的缓冲区个数作了明智的选择。

让我们以说明在某些条件下,从相同的缓冲区圈进行输入和输出的一个有用方法,来结束这一讨论。图 26 介绍了一个新的缓冲区类型,它为紫色。在这种情况下,绿色的缓冲区表示预先读的输入;程序执行 ASSIGN 使一个绿色的缓冲区变成黄色,然后在 RELEASE 时它转成红色,表示有一个块要被输出。输入和输出如同以前一样独立地环绕着缓冲区圈进行,只不过在输出完成之后我们把红的缓冲区变换成紫色,并在输入时把紫色转换成绿色。有必要确保 NEXTG, NEXTR, NEXTP 这些指针都不彼此超越。在图 26 所示的时刻,程序正在 ASSIGN 和 RELEASE 之间进行计算,也在访问黄色

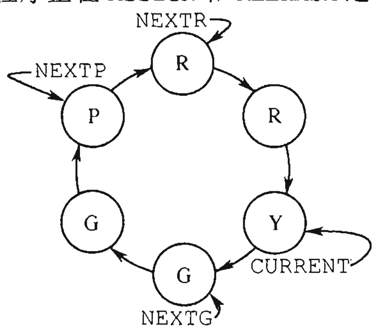


图 26 来自同一个圈的输入和输出

的缓冲区;同时,输入正进入由 NEXTP 所指出的缓冲区;输出则来自由 NEXTR 所指出的缓冲区。

习 题

1. [*05*] (a)如果把 MOVE 指令放置在 JBUS 指令之前,而不是在它之后,序列(3)是否仍然正确? (b)如果把 MOVE 指令放在 IN 命令之后,则将如何?

2. [*10*] 在无缓冲的方式下指令"OUT 1000(6)";"JBUS *(6)"可被用于输出一个带的块,就如指令(1)对输入做这件事一样。通过使用 MOVE 指令和在单元 2000~2099 中的一个辅助缓冲区,试给出对此输出进行缓冲的类似于(2)和(3)的一个方法。

▶3. [*22*] 试编写一个类似于(4)的缓冲区交换的输出子程序。这个子程序叫做 WORDOUT,应把字存入 rA 中作为输出的下一个字,而且如果一个缓冲区满了,它应把 100 个字写到带设备 V 上。变址寄存器 5 应当用来索引当前的缓冲区位置。试说明缓冲区域的布局并说明在程序的开始和结尾需要什么指令(如果有的话)以确保第一个和最后一个块被适当地写出。如果必要,最后的块应补上零。

4. [*M20*] 证明如果一个程序访问单个 I/O 设备,在顺利的情况下,通过 I/O 缓冲我们有可能把运行时间削减一半。但相对于无缓冲的 I/O,我们绝不能减少运行时间多于一半。

▶5. [*M21*] 试把上题的情况推广到程序访问的是 n 个 I/O 设备而不仅仅是一个的情况。

6. [*12*] 应该在一个程序的开始处放置什么指令才能使得 WORDIN 子程序(4)启动于正确的开始处?(例如,变址寄存器 6 必须置为某个值。)

7. [*22*] 试编写一个叫做 WORDIN 的子程序,它除不利用标记外实质上很像(4)。

8. [*11*] 正文叙述了一个假设的输入场景,它由图 23 导致图 24(a),(b)和(c)。在行式打印机的输出正在进行,而不是从卡片输入机的输入正在进行这样一个假设之下,试解释相同的场景。(例如,在图 23 所示的时间正在发生什么事情?)

▶9. [*21*] 导致图 27 中所示的缓冲区内容的一个程序,可以通过下列的时间表来加以表征:

$A,1000,R,1000,A,1000,R,1000,A,1000,R,1000,A,1000,R,1000,$

$A,7000,R,5000,A,7000,R,5000,A,7000,R,5000,A,7000,R,5000,$

$A,1000,R,1000,A,2000,R,1000$。

这个表表示"指定,计算 $1000u$,释放,计算 $1000u$,指定,…,计算 $2000u$,释放,计算 $1000u$",所给出的计算时间不包括计算机可能要等候输出设备赶上来的任何时间间隔(比如在图 27 中的第四个"指定"处)。输出设备以每个块 $7500u$ 的速度运行。

下表表征了随时间的进展图 27 中所示的动作:

时间	动 作	时间	动 作
0	ASSIGN(BUF1)	6000	ASSIGN(等候)
1000	RELEASE,OUT BUF1	8500	BUF1 被指定,OUT BUF2
2000	ASSIGN(BUF2)	9500	RELAESE
3000	RELEASE	10500	ASSIGN(等候)
4000	ASSIGN(BUF3)	16000	BUF2 被指定,OUT BUF3
5000	RELEASE	23000	RELEASE

时间	动 作	时间	动 作
23500	OUT BUF1	59000	RELEASE, OUT BUF2
28000	ASSIGN (BUF3)	64000	ASSIGN (BUF3)
31000	OUT BUF2	65000	RELEASE
35000	RELEASE	66000	ASSIGN(BUF1)
38500	OUT BUF3	66500	OUT BUF3
40000	ASSIGN(BUF1)	68000	RELEASE
46000	输出停止	69000	计算停止
47000	RELEASE,OUT BUF1	74000	OUT BUF1
52000	ASSIGN(BUF2)	81500	输出停止
54500	输出停止		

因此总共需要的时间是 81500u;在 6000 ~ 8500,10500 ~ 16000 以及 69000 ~ 81500 时间范围计算机是空闲的,或者说,总共的空闲时间是 20500u;输出设备在 0 ~ 1000,46000 ~ 47000 以及 54500 ~ 59000 是空闲的,或者说,总共的空闲时间是 6500u。

假定仅有两个缓冲区,试对同一个程序编制类似于上面的时间—动作表。

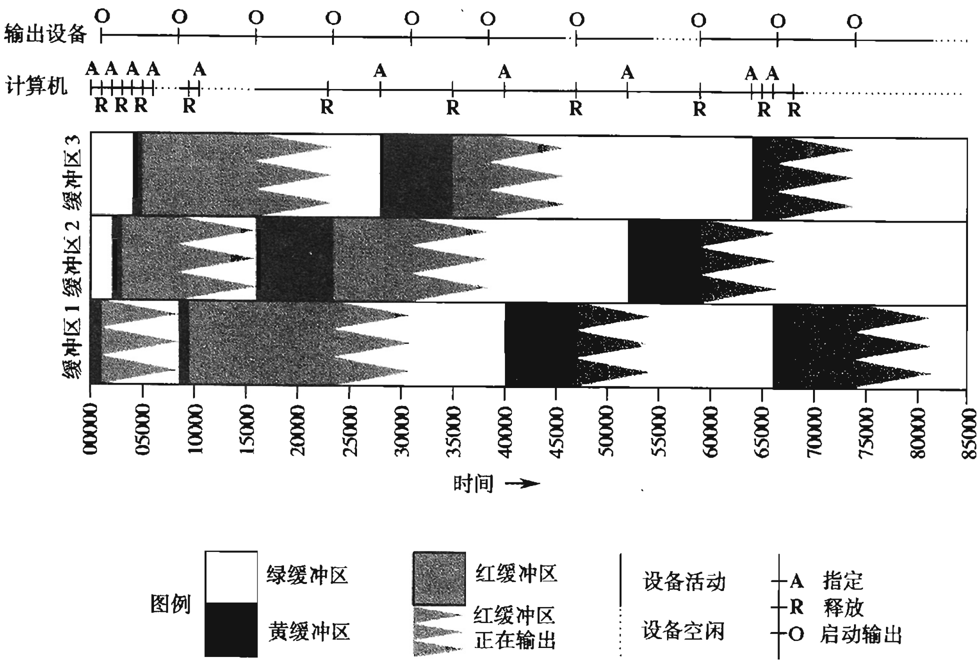


图 27 有三个缓冲区的输出(见习题 9)

10. [*21*] 试使用四个缓冲区,重做习题 9。

11. [*21*] 试仅使用一个缓冲区,重做习题 9。

12. [*24*] 假设把正文中的多缓冲算法用于卡片输入上,并且假设只要在一张卡片的 80 列处读到“.”,则输入就终止。试说明应该如何改变 CONTROL 共行程序(算法 B 和程序 B)使得在此情况下输入被关掉。

13. [*20*] 如果把缓冲算法应用于输出上,则在正文中的 COMPUTE 共行程序的末尾应包括什么指令,以确保所有信息都已从缓冲区输出?

▶ 14. [*20*] 假设计算程序不在 ASSIGN 和 RELEASE 之间交替,而是给出…ASSIGN…ASSIGN…RELEASE…RELEASE 的动作序列。这对于正文中叙述的算法有什么影响?它可能有用吗?

▶ 15. [*22*] 试写出把 100 个块从带设备 0 复制到带设备 1 的一个完整的 MIX 程序,并且只使用三个缓冲区。这个程序应当尽可能地快。

16. [*29*] 以类似于正文中给出的多缓冲的算法的形式,使用三个共行程序(一个控制输入设备,一个用于输出设备,一个用于计算),系统阐述图 26 中提出的"绿 - 黄 - 红 - 紫"算法。

17. [*40*] 把多缓冲区算法修改成缓冲池算法;要在其中建立起使进程避免由于有太多的预先输入而减缓的方法。请尝试使这个算法尽可能精巧。并把你的方法同非建池的方法进行比较,应用于现实的问题上。

▶ 18. [*30*] 一个建议的对 MIX 的扩充允许它的计算被中断,如同下面所说明的那样。你在本题中的任务是修改正文中的算法和程序 A,R 及 B,使得它们使用这些中断设备而不是"JRED"指令。

新的 MIX 的特性包括额外的 3999 个内存单元,即单元 - 3999 到 - 0001。这个机器有两个内部"状态",即正常状态和控制状态。在正常状态下,单元 - 3999 到 - 0001 是不允许访问的内存单元,因而 MIX 计算机像平常一样运行。当出现一个"中断"时,由于后面说明的条件,单元 - 0009直到 - 0001 被设置成等于 MIX 的寄存器的内容,rA 在 - 0009 中;rI1 到 rI6 在 - 0008 到 - 0003中;rX 在 - 0002 中;另有 rJ、溢出开关(OV)、比较指示符(CI)以及下一个指令的地址被保存在 - 0001 中,如下所示:

+	下条指令	OV, CI	rJ

在依赖于中断类型的一个单元处,机器进入控制状态。

单元 - 0010 如同一个"时钟"一样动作:每隔 $1000u$ 的时间,出现在这个单元中的数减 1,当结果为 0 时,则发生转到单元 - 0011 的中断。

新的 MIX 指令"INT"(C = 5, F = 9)工作如下:(a)在正常状态下,发生转到 - 0012 单元的中断。(因此一个程序员可以强迫一个中断出现,以便同一个控制程序进行通信;INT 的地址不起作用,尽管控制程序可以利用它作为区分中断类型的信息。)(b)在控制状态下所有 MIX 的寄存器都从单元 - 0009 到 - 0001 装入,计算机进入正常状态,继续执行。在每种情况下,INT 的执行时间是 $2u$。

在控制状态下给出的 IN,OUT 或 IOC 指令将在 I/O 操作完成时立即引起一个中断。中断转到单元 - (0020 + 设备号)。

而在控制状态时,无中断出现;任何中断条件都被"保存"直到下一个 INT 操作之后,而在正常状态程序的一条指令被实施之后,将发生中断。

▶ 19. [*M28*] 当输入或输出涉及到像磁盘那样的转动设备的短块时,出现特殊条件。假设一个程序对 $n \geqslant 2$ 个连续的信息块以下列方式进行工作:块 k 在时间 t_k 开始被输入,其中 $t_1 = 0$。它在时间 $u_k \geqslant t_k + T$ 时被指定来进行处理并且在时间 $v_k = u_k + C$ 时从它的缓冲区释放出来。这个磁盘每 P 个时间单位转动一圈,而且它的读头每 L 个单位时间就经过一个新的块的起始处;所以必定有 $t_k \equiv (k-1)L \pmod P$。由于处理是串行的,因此必定有,对于 $1 < k \leqslant n$,$u_k \geqslant v_{k-1}$。有 N 个缓冲区,因此对于 $N < k \leqslant n$,$t_k \geqslant v_{k-N}$。

为使得完成时间 v_n 有极小可能的值 $T+C+(n-1)\max(L,C)$，N 要有多大？试给出确定最小的这样的 N 的一般规则。当 $L=1$，$P=100$，$T=.5$，$n=100$，以及 (a) $C=.5$；(b) $C=1.0$；(c) $C=1.01$；(d) $C=1.5$；(e) $C=2.0$；(f) $C=2.5$；(g) $C=10.0$；(h) $C=50.0$；(i) $C=200.0$ 时说明你的规则。

1.4.5 历史和文献

在 1.4 节中叙述的大部分基本技术是由不同的人独立地发展起来的，因此这些思想的确切历史大概将无从得知。在这里我们试图记录那些对历史的最重要贡献，并对它们予以评价。

子程序是为程序员发明的最早的节省劳动的设备。在 19 世纪，Charles Babbage 为他的“分析机器”想像了一个程序库[参见 Philip 和 Emily Morrison 编的*Charles Babbage and His Calculating Engines* (Dover, 1961), 56]；而且，我们可以说，当 Grace M. Hopper 于 1944 年在 Harvard Mark I 上编写了一个计算 sin x 的子程序时，他的理想才变成了现实[参见 *Mechanization of Thought Processes* (London: Nat. Phys. Lab., 1959), 164]。然而，这些实际上是“开放的子程序”，意思是在需要时被插入到一个程序当中，而不是动态地链接。Babbage 所设想的机器是由穿孔卡片序列控制的，如同在 Jacquard 的织布机上那样；Mark I 是由一些纸带控制的。因此它们十分不同于今天的存储程序计算机。

适合于存储程序机器的，并带有作为参数提供的返回地址的子程序链接，是在由 Herman H. Goldstine 和 John von Neumann 于 1946 年和 1947 年所写的广泛传播的程序设计专著中讨论的；参见 Von Neumann 的*Collected Works* **5** (New York: Macmillan, 1963), 215 ~ 235。他们的程序中的主程序，负责把一些参数存入子程序体中，而不是传送必要的信息到寄存器中。在英国，A. M. Turing 早在 1945 年就设计了用于子程序链接的硬件和软件；参见*Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery* (Cambridge, Mass.: Harvard University, 1949), 87 ~ 90；B. E. Carpenter 和 R. W. Doran 编辑，*A. M. Turing's ACE Report of 1946 and Other Papers* (Cambridge, Mass.: MIT Press, 1986), 35 ~ 36, 76, 78 ~ 79。M. V. Wikkes, D. J. Wheeler 和 S. Gill 所写的计算机程序设计的头一本教科书，*The Preparation of Programs for an Electronic Digital Computer* 第 1 版(Reading, Mass.: Addison-Wesley, 1951)的主要课题是一个用途广泛的子程序库的使用和构造。

“共行程序”一词是由 M. E. Conway 于 1958 年创造的，在提出这个概念之后，他还首先把它应用于构造一个汇编程序。大约在同一时期，J. Erdwinn 和 J. Merner 也独立地研究了共行程序。他们写了一篇题为“Bilateral Linkage”的论文，但当时没人认为值得发表，所以很不幸，今天似乎不存在这篇论文的复制品。关于共行程序的概念的头一个公开的阐述，很晚才出现于 Conway 的文章“Design of a Separable Transition-Diagram Compiler”，*CACM* **6** (1963), 396 ~ 408。实际上，共行程序链接的一个原始形式已经在一份早期的 UNIVAC 的出版物[*The Programmer* **1**, 2 (1954 年 2 月), 4]上作为一个“程序设计技巧”被简略地提及。在类似于 ALGOL 的语言中共行程序的适当记号是在 Dahl 和 Nygaard 的 SIMULA I[*CACM* **9** (1966), 671 ~ 678]中引进的，还有好些精彩的共行程序(包括重写的

共行程序)的例子出现在由 O. -J. Dahl,E. W. Dijkstra 以及 C. A. R. Hoare 所著的*Structured Programming* 的第 3 章中。

头一个解释程序可以说是"通用图灵机",即一个有能力模拟任何其它图灵机的图灵机。图灵机不是真正的计算机;它们是用来证明某些问题没有算法解的理论构造。传统意义下的解释程序是由 Mauchly 于 1946 年在 Moore School 的讲课中提到的。早期最著名的解释程序,主要是想提供进行浮点算术运算的方便工具,是用于 Whirlwind I 计算机(C. W. Adams 及其他人编制)和 Illiac I 计算机(D. J. Wheeler 和其他人编制)的某些程序。图灵也参加了这一研究;为 Pilot ACE 计算机使用的解释系统是在他的指导下写出的。有关 20 世纪 50 年代初期解释程序的情况,请参见 J. M. Bennett,D. G. Prinz 和 M. L. Woods 的论文,"Interpretative Sub-routines",*Proc. ACM Nat. Conf.* (1952),81 ~ 87;也请参见由美国华盛顿海军研究所(the Office of Naval Research)出版的*Proceedings of the Symposium on Automatic Programming for Digital Computers* (1954)上的不同文章。

最广泛应用的早期解释程序大概是 John Backus 的"IBM 701 快速编码系统"[请见 *JACM* **1** (1954),4 ~ 6]。这个解释程序由贝尔电话实验室的 V. M. Wolontis 和其他人稍作修改而后又巧妙地重新编写到 IBM 650 上;他们的程序取名"贝尔解释系统",极为流行。由 A. Newell,J. C. Shaw 及 H. A. Simon 于 1956 年开始为十分不同的问题的应用(请见 2.6 节)而设计的 IPL 解释系统,极为广泛地用于表处理上。如同在 1.4.3 小节的引言中提到的,解释程序的现代用法,经常在计算机著作中顺便提到;至于稍微更详细地讨论解释程序的文章,请参见在该小节所开列的参考文献。

头一个跟踪程序是由 Stanley Gill 于 1950 年编制的,可参看他发表在*Proceedings of the Royal Society of London* 上的有趣的文章,A 辑,**206** (1951),538 ~ 554。上面提到的 Wilkes,Wheeler 以及 Gill 的教本,包括有用于跟踪的好些程序。其中最有趣的也许是由 D. J. Wheeler 所写的子程序 C-10,它包括在进入一个库子程序时抑制跟踪,以全速执行子程序,然后继续跟踪的措施。在一般的计算机文献中发表的关于跟踪程序的信息是十分鲜见的,主要是因为所使用的方法都固有地面向特定的机器。就作者所知,仅有的其它早期参考文献是 H. V. Meek 的"An Experimental Monitoring Routine for the IBM 705",*Proc. Western Joint Computer Conf.* (1956),68 ~ 70,它讨论了对一台机器的跟踪程序,而在这台机器上这个问题是特别困难的。现今对于跟踪程序的着重点已经转移到提供选择性的符号输出以及对程序性能的测试的软件上来。这样的系统中佼佼者之一是由 E. Satterthwaite 研制的,并且发表在*Software Practice & Experience* **2** (1972),197 ~ 217 上。

缓冲技术原来是由计算机硬件,以类似于 1.4.4-(3)的代码的方式实现的;不能由程序员访问的内部缓冲区起着单元 2000 ~ 2099 的作用,而在给出了一个输入命令时 1.4.4-(3)的指令隐含在后台执行。在 20 世纪 40 年代末期,对于排序特别有用的软件缓冲技术由 UNIVAC 早期的程序员开发出来(参见 5.5 节)。要全面了解 1952 年人们对 I/O 的流行观点,请参看该年举行的东部计算机联合会议(Eastern Joint Computer Conference)论文集。

DYSEAC 计算机[Alan L. Leiner, *JACM* **1** (1954),57 ~ 81]介绍了在一个程序运行时,I/O 设备直接同内存进行通信并且在通信完成时中断该程序的思想。这样一个系统意

味着已经建立了缓冲的算法,但细节未予公开。在我们所描述的意义下关于缓冲技术头一个公开的文献,给出了一个高度巧妙的方法;参见 O. Mock 和 C. J. Swift 的"Programmed Input-Output Buffering", *Proc. ACM Nat. Conf.* (1958),第 19 篇论文,以及 *JACM* **6** (1959),145 ~ 151。(告诫读者,这两篇文章包含大量地方色彩的专门术语,可能要费些时间才能理解。但是*JACM* **6** 中相近的论文将是有帮助的。)使输入和输出的缓冲得以实现的一个中断系统,是由 E. W. Dijkstra 于 1957 年和 1958 年,在同 B. J. Loopstra 及 C. S. Scholten 的 X-1 计算机的关联中,独立地开发出来的[参见*Comp. J.* **2** (1959),39 ~ 43]。Dijkstra 的博士论文"Communication with an Automatic Computer"(1958),讨论了缓冲技术,在这种情况下它涉及了非常长的缓冲区圈,因为这些程序主要涉及到纸带和打字机的输入输出;每个缓冲区或者包含一个字符或者包含一个数。后来他把这个思想发展成重要的一般的信号灯(semaphore)概念。它对于所有类型的共行过程的控制,而不仅是对于输入输出,都是很基本的[请见 F. Genuys 编的*Programming Languages* (Academic Press,1968),43 ~ 112; *BIT* **8** (1968),174 ~ 186;*Acta Informatica* **1**(1971),115 ~ 138]。David E. Ferguson 的论文"Input-Output Buffering and FORTRAN",*JACM* **7** (1960),1 ~ 9,描述了缓冲区圈并对一次有许多设备的简单缓冲技术给出详细的描述。

About 1,000 instructions is a reasonable upper limit
for the complexity of problems now envisioned.
对于现在能想像到的问题的复杂性来说,
大约 1 000 条指令是合理的上限。
—Herman Goldstine 和 John von Neumann (1946)

第2章 信息结构

I think that I shall never see
A poem lovely as a tree.
我想我永不会见到一首诗
会像一棵树一样可爱。
——JOYCE KILMER (1913)

Yea, from the table of my memory
I'll wipe away all trivial fond records.
是的,我要从我的记忆中
抹去我喜爱的一切微不足道的印象。
——Hamlet (Act I, Scene 5, Line 98)

2.1 引论

计算机程序通常都是对一些信息表进行操作。在大多数情况下,这些表并不仅仅是杂乱无章的数值集团;它们含有数据元素之间重要的结构关系。

在最简单的形式下,一个表可以是元素的一个线性表。这时它的有关结构的性质可以包括对诸如下列这样一些问题的答案:哪个元素是表中的头一个元素?哪个元素是最后的元素?哪些元素居于给定的一个元素之前和之后?在这种个表中有多少个元素?即使在这种看起来简单的情况下,关于结构也大有可谈的问题(请见2.2节)。

在更复杂的情况下,表可以是一个二维的数组(一个矩阵或栅格,有一个行和一个列的结构),或者是具有更高 n 值的 n 维数组;它可以是一个树结构,表示层次或分支关系;或者是复杂的具有大量交互联系的多重链接结构,如同在人的大脑中我们可以找到的那样。

为了合理地使用一台计算机,我们需要理解存在于数据内的结构关系,以及在一台计算机内表示和操作这样的结构的基本技术。

本章综述关于信息结构最重要的事实:不同类型结构的静态和动态的性质,进行存储分配的手段和结构化数据的表示,以及建立、改变、存取和破坏结构信息的有效算法。在进行这一研究的过程中,我们还将给出若干重要的例子,来说明这些方法对于各种各样问题的应用。这些例子包括拓扑排序、多项式的算术运算、离散系统模拟、稀疏矩阵变换、代数公式的操作以及对编写编译程序和操作系统的应用。我们所关心的几乎完

全是在一台计算机内部所表示的结构;从外部到内部表示的转换将是第 9 章和第 10 章的课题。

我们将讨论的许多材料通常叫做"列表(List)处理",因为人们已设计了像 LISP 这样一些程序设计系统,以方便于对称做列表的一般类型结构的工作。(当在这一章用"List"一词时,它表示 2.3.5 小节详加研究的一种特殊结构)*。尽管列表处理系统对于大量的情况都是有用的,但它们对程序员附加了通常并非必需的限制;通常最好是在自己的程序中直接使用本章的方法剪裁数据格式和处理对于特定应用的算法。不幸的是许多人仍然觉得列表处理技术十分复杂(以致有必要使用由他人精心写成的解释系统或者一个预先编制的子程序集合),并且觉得仅仅在某种固定的方式下,才必须使用列表处理。我们将看到,对于处理复杂结构的方法没有什么有魔力的,神秘的和困难的工作;这些技术是每个程序员的"武库"中的一个重要部分。因此无论是在用汇编语言还是使用像 FORTRAN,C 和 Java 这样的代数语言写程序,我们都能容易地使用它们。

我们将借助于 MIX 计算机来说明处理信息结构的一些方法。不打算仔细考察详细的 MIX 程序的读者至少应该学习在 MIX 的内存中表示结构信息的一些方法。

在这里重要的是定义今后将经常使用的若干术语和记号。一张表中的信息由一组**节点**(有些作者称之为"记录","实体"或"珠")组成;有时我们将称之为"项"或"元素"以替代"节点"。每个节点由计算机内存的一个或多个连续的字组成,它们被分成为称做**字段**的命名了的部分。在最简单的情况下,一个节点只不过是一个内存的字,或者它只是由该整个字组成的一个字段。作为一个更有趣的例子,假设我们打算用表格的元素来表示扑克牌;我们可以有分成为五个字段——TAG,SUIT,RANK,NEXT 以及 TITLE 的两个字的节点

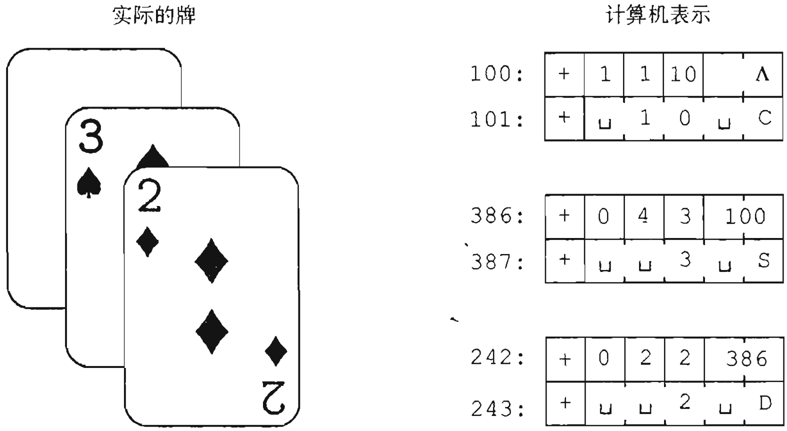
+	TAG	SUIT	RANK	NEXT
+	TITLE			

(1)

(这个格式反映了两个 MIX 字的内容。回忆一下,一个 MIX 字由五个字节和一个符号组成;参见 1.3.1 小节。在这个例子中,我们假定在每个字中的符号是 +。)一个节点的**地址**,也叫做一个**链接**、**指针**或该节点的**引用**,是它的头一个字的内存单元。地址通常相对于某个基地址来取。但在本章中为简便起见,我们将把地址取成为一个绝对的内存单元。

在一个节点内任何字段的内容可以表示数、字母、链接或者程序员可能要的任何其它东西。同上述例子相关联,我们可能希望表示可以出现在单人纸牌游戏中的一组牌;TAG = 1 表示这张牌是面朝下的,而 TAG = 0 表示牌是面朝上的;SUIT = 1,2,3 或 4 分别表示梅花、方片、红桃或黑桃;RANK = 1,2,…,13 表示一点,两点,…,老 K;NEXT 是对这组牌中这张牌下边的牌的链接;而 TITLE 是供打印时使用的这张牌的五个字符的名字。典型的一组牌看起来像下面这样:

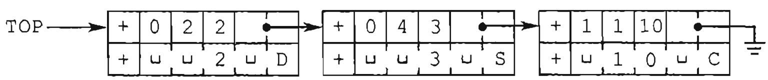
* List 统译为列表,而 list 译为表。——译者注



(2)

在计算机表示中的内存单元在这里被示作 100,386 及 242;就这个例子而言它们可以是任何其它的数,因为每张牌都链接到它下面的那张牌。注意在节点 100 中的特殊链接"Λ";*我们使用大写希腊字母 Λ 来表示空链接*,即对于无节点的链接。空链接 Λ 出现在节点 100 中因为梅花 10 是这组牌中最下面的牌。在这个机器内,Λ 用不可能是一个节点地址而可以容易地识别的某个值来代表。我们一般假定,没有节点出现在 0 单元中;结果,Λ 就几乎总是被表示为 MIX 程序中的链接值 0。

到数据的其它元素的链接的引入是计算机程序设计中一个极其重要的思想;链接是表示复杂结构的关键。当图示节点的计算机表示时,通常通过箭头来表示链接是很方便的,这样(2)将呈现为



(3)

实际的单元 242,386 和 100(它们是无关紧要的)不再出现于表示(3)中。一个"接地"线的电路记号用于表示一个空链接,在这里被示于图的右边。还注意(3)通过由"TOP"出发的箭头来表示顶部的牌;这里 TOP 是一个通常称做指针变量的链接变量,即其值是一个链接的变量。对一个程序中节点的所有访问都被做成直接地通过链接变量(或链接常数)或者间接地通过在其它节点的链接字段进行。

现在我们进到记号的最重要部分,即访问节点内字段的方法。这简单地通过给出字段名,后面接上在括弧中的对所需要节点的一个链接来完成;例如在(1),(2)和(3)中我们有

RANK(100) = 10;　SUIT(TOP) = 2
TITLE(TOP) = "␣␣2␣D";　RANK(NEXT(TOP)) = 3　(4)

读者应仔细研究这些例子,因为这样的字段记号将被使用于本章和随后各章的许多算法中。为使这些思想更加清晰,我们现在将叙述用于把一张面朝上的牌放到这堆牌的顶部的一个简单算法,并假定 NEWCARD 是一个链接变量,其值为指向新牌的链接。

A1. 置 NEXT(NEWCARD)←TOP。(这把适当的链接放进新牌节点中。)
A2. 置 TOP←NEWCARD。(这使 TOP 保持指向这组牌的顶部。)
A3. 置 TAG(TOP)←0。(这把这张牌标记为面朝上。) ▌

另一个例子是以下的算法,它累计当前这组牌中牌的张数:

B1. 置 N←0,X←TOP。(这里 N 是一个整数变量,X 是一个链接变量。)
B2. 如果 X = Λ,则停止;N 是这组牌中牌的张数。
B3. 置 N←N + 1,X←NEXT(X),并转回步骤 B2。 ▌

注意在这些算法中我们对两个十分不同的事情使用了符号名,即变量名(TOP,NEWCARD,N,X)和字段名(TAG,NEXT)。不应该混淆这两个用法。如果 F 是一字段名和 L≠Λ 是一个链接,则 F(L)是一个变量;但 F 本身不是一个变量——它不具有一个值,除非它是通过一个非空链接来加以定性。

当讨论低级的机器细节时,我们还使用另外两个记号,以便在地址和存储在那里的值之间进行转换。

a)CONTENTS 总是表示一个单字节点的全字字段。因此 CONTENTS(1000)表示存在内存单元 1000 中的值;它是有这个值的一个变量。如果 V 是一个链接变量,则 CONTENTS(V)表示由 V 所指的值(不是 V 本身的值)。

b)若 V 是在一个内存单元中保存的某个值的名称,则 LOC(V)表示该单元的地址。因此,若 V 是一个变量,其值被保存在内存的一个全字中,我们有 CONTENTS(LOC(V)) = V。

尽管 MIXAL 的记号是稍微落后了,但是把这个记号转换成 MIXAL 汇编语言却很容易。可以把链接变量的值放进变址寄存器中,并使用 MIX 的表达部分字段的能力来索引所需要的字段。例如可以把上面的算法 A 写成如下这样:

```
NEXT   EQU   4:5          对于汇编程序 NEXT 字段
TAG    EQU   1:1            和 TAG 字段的定义
       LD1   NEWCARD      A1. rI1←NEWCARD
       LDA   TOP            rA←TOP                      (5)
       STA   0,1(NEXT)      NEXT(rI1)←rA
       ST1   TOP          A2. TOP←rI1
       STZ   0,1(TAG)     A3. TAG(rI1)←0      ▌
```

在一台计算机中可以很容易和很有效率地进行这些运算,这是"链接存储单元"概念具有重要性的主要原因。

有时,我们有表示一个全节点的单个变量;它的值是一个字段序列而不仅仅是一个字段。因此我们可以写

$$\text{CARD}\leftarrow\text{NODE}(\text{TOP}) \tag{6}$$

其中 NODE 就像 CONTENTS 一样是一个字段说明,只不过它指的是整个节点,而且其中 CARD 是一个具有(1)那样结构化值的变量。如果在一个节点中有 c 个字,记号(6)是对于下列 c 个低级赋值的缩写:

$$\text{CONTENTS(LOC (CARD)} + j) \leftarrow \text{CONTENTS(TOP} + j),\ 0 \leq j < c \tag{7}$$

在汇编语言和用于算法中的记号之间有一个重要的区别。由于汇编语言是接近于机器内部语言的，用于 MIXAL 程序中的符号表示地址而不是值。因此，在(5)的左边的列中，符号 TOP 实际上表示指向顶部纸牌的指针在内存中的地址；但是在(6)和(7)以及在(5)右边的注解当中，它表示 TOP 的值，即顶部纸牌节点的地址。汇编语言和高级语言之间的这个差别经常会被程序员新手混淆；因此我们鼓励读者来做习题 7。其它习题也提供了本节所介绍的记号约定的有用训练。

习　题

1. [*04*] 在(3)所描述的情况下，什么是(a) SUIT(NEXT(TOP))；(b) NEXT(NEXT(NEXT(TOP)))的值？

2. [*10*] 正文中指出，在许多情况下 CONTENTS(LOC(V)) = V。在什么条件下，我们有 LOC(CONTENTS(V)) = V？

3. [*11*] 给出一个实际上解除算法 A 的效果的算法：它去掉这组牌顶上的牌(如果这组牌不空的话)，而且把 NEWCARD 置为这张牌的地址。

4. [*18*] 给出类似于算法 A 的一个算法，不同的是它把新的一张牌面朝下地放在这组牌的底部(这组牌可能是空的)。

▶ 5. [*21*] 给出实际上解除问题 4 的效果的一个算法；假定这组牌不空，而且它底部的牌面朝下。你的算法应当去掉底部的牌，而且使 NEWCARD 链接到它。(这个算法在单人纸牌游戏中有时叫做“骗牌”。)

6. [*06*] 在玩纸牌的例子中，假设 CARD 是一个变量名，其值如同在(6)中那样是整个节点。操作 CARD←NODE(TOP)把 CARD 的诸字段分别置成等于这组牌顶部的相应值。在这个操作之后，下列记号的哪些代表顶部的牌的花色：(a) SUIT(CARD)；(b) SUIT(LOC (CARD))；(c) SUIT(CONTENTS(CARD))；(d) SUIT(TOP)？

▶ 7. [*04*] 在正文的例子 MIX 程序(5)中，链接变量 TOP 被保存在其汇编语言是 TOP 的计算机字中。给定字段结构(1)，下列编码序列的哪一个把数量 NEXT(TOP)放进寄存器 A 中？说明为什么另一个序列是不正确的？

a)
```
LDA TOP(NEXT)
```
b)
```
LD1   TOP
LDA   0,1(NEXT)
```

▶ 8. [*18*] 写出对应于算法 B 的一个 MIX 程序。

9. [*23*] 写出由顶部的牌开始，打印出一组牌的当前内容的字符名的一个 MIX 程序，而且每行打印一张牌，对于面朝下的牌用圆括号括起来。

2.2 线性表

2.2.1 栈、队列和双端队列

出现在数据中的结构信息，比起我们真正直接地要在一台计算机中表示的，通常要多得多。例如，在上一节中的每张“扑克牌”节点中都有一个 NEXT 字段来确定在这组牌中什么牌在它下面，但是我们没有提供直接的方法来找出什么牌(如果有的话)，是在一张给定纸牌的上面，或者找出一张给定的纸牌是在哪一组当中。而且当然我们全然隐藏了实际的扑克牌的大多数特性：纸牌背面的设计细节；在玩牌的房间里，纸牌和其它对象的关系，组成这些牌的各个分子，等等。可以想像，这样的结构化信息与某些计算机应用有关系，但是显然，我们并不需要存储出现在每种情况下的所有结构。确实，对于大多数玩牌的情况，我们将不需要保留在前边例子中的所有事实；TAG 字段，它指出一张牌是面朝上还是面朝下，通常将是不需要的。

我们在每种情况下都必须决定有多少结构要表示在我们的表中；以及如何使每个信息都可访问。为做出这样的判断，我们需要知道对数据要执行什么操作。因此，对于在本章中考虑的每个问题，*我们不仅考虑数据结构，而且还要考虑对于数据要进行的操作种类*；计算机表示的设计依赖于对数据所要求的功能，以及它的固有性质。确实，对功能与形式的强调一般说来对于设计问题是很基本的。

为了进一步说明这一点，我们考虑计算机硬件设计的一个相关方面。一台计算机的内存通常被分类为“随机存取存储器”，如同 MIX 的主存储器那样；或者作为“只读存储器”，它应当包含实际上不变的信息；或者一个“辅助的海量存储器”，像 MIX 的磁盘设备，尽管它可存储大量的信息，但是却不能以高速来对它进行访问；或者一个“关联存储器”，更适当地说，它应叫做“按内容编址的存储器”，对于它来说，信息是通过它的值而不是通过它的地址来访问的，等等。每种存储器意想中的功能是如此重要，因而对它赋予具体存储器类型的名字；所有这些设备都是“存储器”设备，但是设置它们的目的深远地影响它们的设计和它们的价格。

一个线性表是 $n \geqslant 0$ 个节点 X[1], X[2], …, X[n]的一个序列，当这个序列的所有项出现在一行中时，这个序列的实质性的结构性质仅仅涉及这些项之间的相对位置。对于这样一个结构，我们关心的惟一的事情是，如果 $n > 0$，X[1]是头一个节点而 X[n]是最后的节点；而如果 $1 < k < n$，则第 k 个节点是居于 X[$k-1$]之后而居于 X[$k+1$]之前。

对线性表我们可能要实施的运算包括例如下面这些：

i)访问表的第 k 个节点以考察和/或改变它的字段的内容。

ii)在第 k 个节点之前或之后插入一个新节点。

iii)删去第 k 个节点。

iv)把两个或更多的线性表组合成一个表。

v)把一个线性表分成两个或更多的表。

vi)复制一个线性表。

vii)确定在一个表中的节点个数。

viii)基于节点的某些字段把表的节点排成递增顺序。

ix)在表中查找在某个字段中具有特定值的一个节点。

在 i),ii)和 iii)的操作中,特殊情况 $k=1$ 和 $k=n$ 是具有头等重要性的,因为一个线性表的头一项和最末一项可能要比一般的项更易于得到。在本章中我们将不讨论操作 viii)和 ix),因为这些课题分别是第 5 章和第 6 章的主题。

在一般情况下,一个计算机应用很少要求所有这九种操作,因此我们感到,根据最经常地进行的操作的种类,有许多方式来表示线性表。对于线性表很难设计一种单一的表示方法,以使其中所有的这些操作都是高效率的;例如,如果在表的中间进行插入和删去项的操作的同时,要对随机的 k 来访问一个长的表的第 k 个节点,是比较困难的。因此,我们依据要施行的主要操作,来区分线性表的种类,就如同已经说明过的那样,计算机的存储器是按它们的应用来分类的。

我们很经常遇到的线性表几乎总是在它的头一个节点或最后一个节点进行插入、删除或访问值的操作,因此我们给它们特殊的名字:

栈是所有插入和删除(而且通常所有的访问亦然)都在表的一端进行的一种线性表。

队列是所有的插入在表的一端进行,而所有的删除(而且通常所有的访问亦然)在另一端进行的一种线性表。

双端队列是所有的插入和删除(而且通常所有访问亦然)在表的两端进行的一种线性表。

因此双端队列比栈或队列更为一般;它和一组牌有某些相同的性质,其表述也以同样的方式来进行。我们也区分输出受限双端队列和输入受限双端队列,即分别地仅仅在一端允许删除和插入。

在某些学科中,"队列"一词在广得多的意义下使用,以描述要进行插入和删除的任何种类的表。于是上面标出的特殊情况便称做各种"排队规则"。然而,本书只打算通过模拟人们排成行等候得到某种服务的有序队列,有限制地使用"队列"这一术语。

有时,如同 E. W. Dijkstra 所建议的那样(见图 1),借助于同铁路车辆的转向的类比,来理解一个栈的机制是有帮助的。图 2 示出对于双端队列相应的图形。

对于一个栈,我们总是删去当前在表中"最新的"项,即比任何项都要更晚才被插入的项。对于一个队列,则反过来,总是删去"最老的"项;节点离开表的顺序和它们进入的顺序相同。

许多认识到栈和队列重要性的人,曾独立地对它们赋予了不同的名字:栈被叫做"下推表",反向存储器,窖(cellars),嵌套存储器,堆(piles),后进先出(LIFO)表,甚至叫做哟哟(yo-yo)表。队列有时叫做循环存储或先进先出("FIFO")表。会计人员已经使用后进先出和先进先出这些词多年,作为打印库存目录的方法的名字。另外还有一个术语"架子"(shelf),被应用于输出受限双端队列,而输入受限队列被称做"卷形物"(scroll 或 roll)。名称的多样性本身就很有趣,因为它是这些概念的重要性的证据。栈和队列这

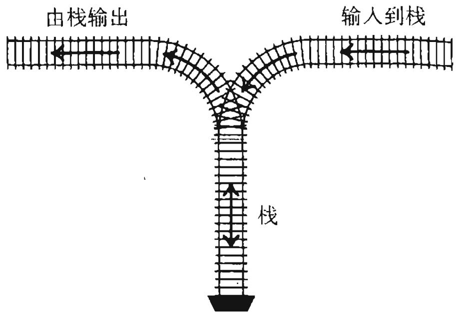


图 1 一个表示为铁路转轨网络的栈

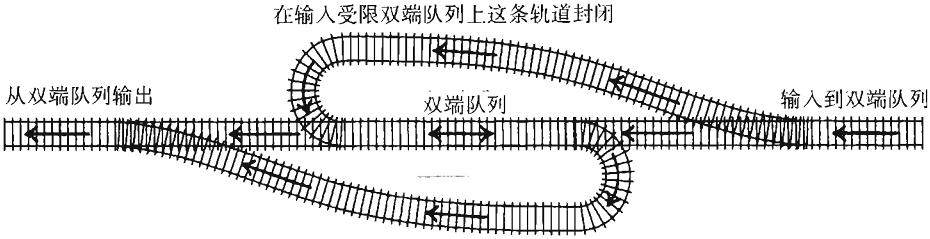
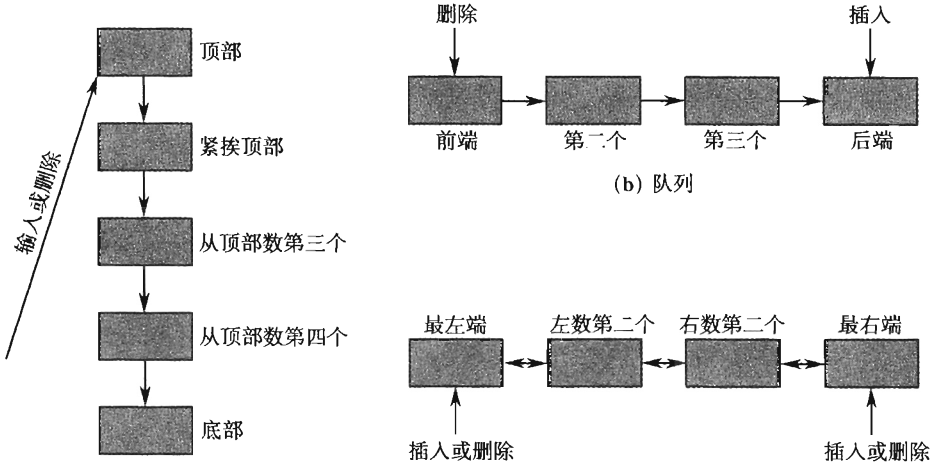


图 2 一个表示为铁路转轨网络的双端队列

两个词已逐渐变成标准术语;在上面所列举的其它词中,只有"下推表"还相当流行,特别是在同自动机理论相关联时。

实践中栈十分经常地出现。例如,我们可以处理一组数据并且编制一张关于异常条件或稍后要做的事情的清单;在处理完原始数据之后,再回到这个清单来,我们可以做剩下的处理,删去这些表项直到它变成空为止。("鞍点"问题,即习题 1.3.2-10,是这种状况的一个实例。)对于这样一个清单,栈或者队列都将是合适的。但栈一般来说更为方便。当我们解决下列这样的问题时,我们总是在心里想着"栈":一个问题导致另一个问题,而它又导致另一个问题;我们把问题和子问题压入栈中,并且在解决了它们之后就把它们从栈中删去。类似地,在执行一个计算机程序时,进入和离开子程序的过程也有一个类似栈的行为属性。对于处理具有嵌套结构的语言,如程序设计语言,算术表达式以及德文的"嵌套句"(Schachtelsätze)等说来,栈特别有用。一般地说,栈最经常同明显的或隐含的递归算法有关,因此我们将在整个第 8 章中来讨论这一联系。

当算法引用这些结构时,一般使用特殊的术语:我们把一项压入栈的*顶部*,或者把一项从栈的顶部弹出(见图 3(a))。栈的*底部*是最少访问的项,在所有其它项都被删除之前它将不被删除。(人们通常说,他们把一项*压入*栈,以及当栈顶部元素被删除时说*弹出*栈。这个术语来自于在自助餐馆里通常看到的盘碟摞起的类似情况,"压入"和"弹出"这些词的简洁性有其优点,但是这些术语却错误地隐喻了在计算机存储器中整个表的移动。在物理上说,没有什么被压下(或下推);把项加到栈顶上,同堆干草或摞箱子



(a) 栈　(b) 队列　(c) 双端队列

图 3　线性表的三个重要类

是一样的。)对于队列,我们说起队列的*前端*和*后端*;事物进入后端,而且当它们最终达到前端的位置时被删去(见图 3(b))。当谈及双端队列时,我们说起*左端*和*右端*(图 3(c))。顶部、底部、前端与后端的概念有时也应用于用作栈或队列的双端队列中,而且没有标准的约定是否顶部、前端和后端应当出现在左边还是右边。

因此在我们的算法中可以方便地使用来自英语的丰富多样的描述性词汇:对于栈使用"上下"的术语,对于队列使用"在队列中等候"的术语,对于双端队列使用"左右"的术语。

为处理栈和队列,已经证明再增加一点附加的记号是方便的:我们写

$$A \Leftarrow x \tag{1}$$

(当 A 是一个栈时)表示把值 x 插入到栈 A 的顶部或者(当 A 是一个队列时)表示把 x 插入到队列的尾部。类似地,记号

$$x \Leftarrow A \tag{2}$$

用来表示把变量 x 置成等于位于栈 A 顶部的值或位于队列 A 前端的值,而且从 A 中删除这个值。当 A 为空——即当 A 不包含值时,记号(2)无意义。

如果 A 是非空的栈,我们可以写

$$\mathrm{top}(A) \tag{3}$$

来表示它的顶部元素。

习　题

1. [*06*] 一个输入受限的双端队列是项可以插入到其一端,而项的删除则在两端进行的线性表;显然,如果我们始终从两端之一来删除所有项,一个输入受限的双端队列可以作为一个栈

或者一个队列来操作。一个输出受限的双端队列也能作为栈或队列操作吗？

▶ 2. [*15*] 想像四列列车位于图 1 中轨道的输入一边，从左至右编号为 1,2,3 和 4。假设我们实现下列操作(它同图中所给出的箭头的方向一致，而且不允许车辆"跳越"其它车辆)：(i)将车 1 移入栈中；(ii)将车 2 移入栈中；(iii)把车 2 移到输出端；(iv)把车 3 移入栈中；(v)把车 4 移入栈中；(vi)把车 4 移到输出端；(vii)把车 3 移到输出端；(viii)把车 1 移到输出端。

作为这些操作的结果，原来车辆的顺序 1234 现在已改变成 2431。*本题和以下诸题的目的是考察以这样的方式，从栈、队列和双端队列可得到什么排列。*

如果有编号为 123456 的六列列车，它们能否被排列成顺序 325641？它们能否被排列成 154623？(如果能，示出是怎么做的。)

3. [*25*] 上一道题中的操作(i)到(viii)可以更简明地以编码 SSXSSXXX 来加以描述，其中 S 表示"把一列车从输入移动到栈中"，而 X 表示"把一列车从栈移动到输出端"。某些 S 和 X 的序列确定无意义的操作，因为在所指定的轨道上可能没有车辆；例如，序列 SXXSSXXS 不能实现，因为我们假定栈开始时是空的。

如果一个包含 n 个 S 和 n 个 X 的序列不指定不能被实现的操作，我们称该序列是*允许的*。试系统地阐述一个规则，由这个规则，容易区分允许和不允许的序列；进一步说明，两个不同的允许的序列不会给出相同的输出排列。

4. [*M34*] 求出通过习题 2 那样的栈，可以得到的 n 个元素的排列个数 a_n 的简单公式。

▶ 5. [*M28*] 证明利用一个栈从 $1\ 2\ \cdots\ n$，有可能得到一个排列 $p_1\ p_2\ \cdots\ p_n$ 当且仅当没有下标 $i<j<k$ 存在使得 $p_j<p_k<p_i$。

6. [*00*] 考虑以一个队列代替栈之下的习题 2 的问题，通过使用一个队列可以得到 $1\ 2\ \cdots\ n$ 的什么排列？

▶ 7. [*25*] 考虑以一个双端队列代替栈之下的习题 2 的问题。(a)求 1 2 3 4 的一个排列，它能够通过一个输入受限的双端队列得到，但不能通过输出受限的双端队列得到。(b)求通过输出受限的双端队列可以得到但通过输入受限的双端队列不可能得到的 1 2 3 4 的一个排列。[从(a)和(b)的结果可知，在输入受限和输出受限的双端队列之间肯定有差别。](c)求通过输入受限的双端队列不可能得到，通过输出受限的双端队列也不可能得到的 1 2 3 4 的一个排列。

8. [*22*] 是否有使用既非输入受限也非输出受限的双端队列不可能得到的 $1\ 2\ \cdots\ n$ 的排列？

9. [*M20*] 设 b_n 是通过使用输入受限的双端队列可以得到的 n 个元素的排列的个数。(注意，如习题 7 所示，$b_4=22$。)证明，b_n 也是通过使用输出受限的双端队列可以得到的 n 个元素的排列的个数。

10. [*M25*] (见习题 3) 设 S, Q 和 X 分别表示在左边插入一个元素，在右边插入一个元素，以及从一个输出受限的双端队列的左边取出一个元素的操作。例如，序列 QQXSXSXX 将把输入序列 1 2 3 4 转换成为 1 3 4 2。序列 SXQSXSXX 给出相同的转换。

试求出定义符号 S, Q 和 X 的*允许的*序列的概念的一个方法，使得下列性质成立：对于一个输出受限的双端队列，可得到的 n 个元素的每一个排列精确地对应一个允许的序列。

▶ 11. [*M40*] 作为习题 9 和 10 的结果，数 b_n 是长度为 $2n$ 的允许序列的个数。试求对于生成函数 $\sum_{n\geq 0} b_n z^n$ 的一个封闭形式。

12. [*HM34*] 计算在习题 4 和 11 中的量 a_n 和 b_n 的近似值。

13. [*M48*] 通过使用一般的双端队列可以得到多少个 n 个元素的排列？[关于在 $O(n)$ 步之内判定一个给定的排列是否可以得到的算法，参见 Rosenstiehl 和 Tarjan, *J. Algorithms* **5**(1984),

389 ~ 390。]

▶ 14. [*26*] 假设只允许使用栈作为数据结构，如何能有效地以两个栈实现一个队列？

2.2.2 顺序分配

在一台计算机内保存一个线性表的最简单和最自然的方法是把表项放进连续的单元中，即一个挨一个地放。于是我们将有

$$\mathrm{LOC}(\mathrm{X}[j+1]) = \mathrm{LOC}(\mathrm{X}[j]) + c$$

其中 c 是每个节点的字数。(通常 $c=1$。当 $c>1$ 时，有时把一个表分成为 c 个“平行”的表更为方便，使得依赖于 k，节点 X[j]的第 k 个字被保存在从 X[j]的头一个字的单元到一个固定的距离处。然而，我们将继续假定，这些相邻的由 c 个字组成的字组，都形成一个单个的节点。)一般地，

$$\mathrm{LOC}(\mathrm{X}[j]) = \mathrm{L}_0 + cj \tag{1}$$

其中 L_0 是称为基地址的常数，即人为地假定的节点 X[0]的单元。

这一表示线性表的技术如此明显和尽人皆知，因此似乎不需要再对它费任何口舌。但是，我们将在本章的稍后看到许多其它“更复杂的”表示方法，因而首先考察简单情况看看由之我们可以走多远，是一个好想法。重要的是既要理解使用顺序分配的效力，也要弄清楚其局限。

顺序分配对于处理栈来说是十分方便的。我们简单地设定一个称为栈指针的变量 T。当栈为空时，令 T = 0。为了把一个新元素 Y 放到栈的顶部，我们置

$$\mathrm{T} \leftarrow \mathrm{T} + 1;\quad \mathrm{X}[\mathrm{T}] \leftarrow \mathrm{Y} \tag{2}$$

而且当栈非空时，我们可以把 Y 置成等于顶部节点以及通过逆转(2)的操作删去该节点：

$$\mathrm{Y} \leftarrow \mathrm{X}[\mathrm{T}];\quad \mathrm{T} \leftarrow \mathrm{T} - 1 \tag{3}$$

(在一台计算机内部，因为式(1)的缘故，维持 cT 的值而不是 T 的值通常是最有效率的。这样的修改很容易做，因此我们将继续按 $c=1$ 进行我们的讨论。)

一个队列或者更一般的双端队列的表示是有些技巧的。一个明显的解决方法是维持两个指针，比如说 F 和 R(表示队列的前端和后端)，且当队列为空时 F = R = 0。于是把一个元素插入队列的后端将是

$$\mathrm{R} \leftarrow \mathrm{R} + 1;\ \mathrm{X}[\mathrm{R}] \leftarrow \mathrm{Y} \tag{4}$$

删去前端节点(F 刚好指前端之下)将是

$$\mathrm{F} \leftarrow \mathrm{F} + 1;\ \mathrm{Y} \leftarrow \mathrm{X}[\mathrm{F}];\ \text{如果 } \mathrm{F} = \mathrm{R}, \text{则置 } \mathrm{F} \leftarrow \mathrm{R} \leftarrow 0 \tag{5}$$

但注意会发生什么情况：如果 R 总是居于 F 之前(使得在队列中总是至少有一个节点)，则所用的表项是 X[1], X[2], …, X[1000], …，直至无穷，因而这是对于存储空间极其糟糕的浪费。因此(4)和(5)的简单方法只应当在已知 F 经常赶上 R——例如，如果一下子有很多删除操作把队列清空时，才加以使用。

为了避免队列过多地占用存储器的问题，我们可以把 M 个节点 X[1], X[2], …, X[M]暗地里地放置在一个圆圈上且使 X[1]紧挨 X[M]。上面的过程(4)和(5)变成

如果 R=M,则 R←1,否则 R←R+1;X[R]←Y (6)

如果 F=M,则 F←1,否则 F←F+1;Y←X[F] (7)

事实上,当我们观察 1.4.4 小节中的输入输出缓冲时,我们已经看见像现在这样一个循环的队列。

我们迄今为止的讨论是非常不实际的,因为我们暗中假定,什么东西也不出错。当从一个栈或队列删除一个节点时,我们假定至少有一个节点存在。当向一个栈或队列插入一个节点时,我们假定在存储器中有供存放它的空间。但是很清楚(6)和(7)的方法允许在整个队列中至多有 M 个节点,而方法(2),(3),(4),(5)只允许 T 和 R 达到在任何给定的计算机程序内的一个确定的极大数量。下列描述说明在我们不假设这些限制自动地满足的普通情况下怎样重写这些动作:

X⇐Y(插入栈): { T←T+1; 如果 T>M 则 OVERFLOW(上溢); X[T]←Y } (2a)

Y⇐X(从栈中删除): { 如果 T=0,则 UNDERFLOW(下溢); Y←X[T]; T←T-1 } (3a)

X⇐Y(插入队列): { 如果 R=M,则 R←1,否则 R←R+1; 如果 R=F,则 OVERFLOW; X[R]←Y } (6a)

Y⇐X(从队列中删除): { 如果 F=R,则 UNDERFLOW; 如果 F=M,则 F←1,否则 F←F+1; Y←X[F] } (7a)

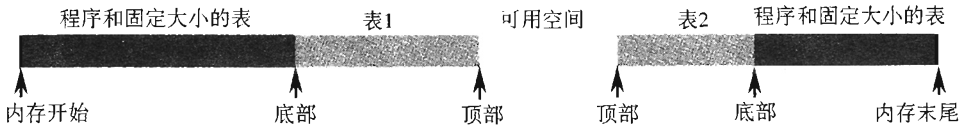
这里我们假定 X[1],…,X[M]是表允许的总共的空间数量;OVERFLOW 和 UNDERFLOW 指项的富余或不足。当我们使用(6a)和(7a)时对队列指针的初始设定 F=R=0 不再是正确的,因为当 F=0 时溢出将不被检测出来;我们应当以比如说 F=R=1 开始。

鼓励读者做习题 1,它讨论这种简单的队列机制的一个非平凡的方面。

下一个问题是:"当出现 UNDERFLOW 或 OVERFLOW 时,我们做什么?"在 UNDERFLOW 的情况下,我们已经试图删去一个不存在的项;这通常是一个有意义的条件——不是一个出错的情况——它可以用来支配一个程序的流动。例如,我们可能要重复地删除项直到出现 UNDERFLOW 为止。然而,OVERFLOW 的情况通常是一个错误;它意味着表格已经满了,但仍然有等候被放入的更多的信息。在 OVERFLOW 情况下通常的政策是不情愿地报告,由于程序的存储能力已被超过,因此程序不能继续执行;于是程序终止。

当只有一个表变成太大,而同一个程序的其它表可能还剩有大量空间可用时,我们当然不高兴在 OVERFLOW 的情况下就放弃了。在上面的讨论中,我们主要是考虑仅有一个表的程序。然而,我们经常遇到涉及好些个栈的程序,每个栈的大小都是动态变化的。在这样一种情况下,我们不要对每个栈强加一个极大的容量,因为这个容量通常是不可预测的;而且即使对于每个栈已经确定一个极大的容量,我们将很少发现所有栈同时充满它们的极大容量。

当恰有两个大小可变的表时，如果我们令这两个表彼此相向地增长，则它们可以很好地共存：



这里表 1 向右扩展，而表 2(以相反方向存储)向左扩展。除非两个表的总容量穷尽所有的存储空间，否则 OVERFLOW 将不出现。这些表可以独立地扩展和收缩使得每一个的最大有效容量都要大大超过可用空间的一半。存储空间的这种布局使用很频繁。

然而我们可以很容易地使自己相信，没有办法在内存中存储三个或更多的可变大小的顺序表，使得(a)仅当所有表的总容量超出总的空间时，才出现 OVERFLOW，以及(b)每个表的“底部”元素都放在一个固定的单元中。比如说，当有十个或更多个可变大小的表——而这并非不寻常——时，存储分配问题变得非常重要。如果我们希望满足条件(a)，就必须放弃条件(b)，即必须允许这些表的“底部”元素改变它们的位置。这意味着，等式(1)的单元 L_0 不再是常数；由于所有的访问都是相对于基地址 L_0 进行的，因此对于表的访问也就不能做成是对于一个绝对的存储地址进行的。在 MIX 的情况下，把第 I 个一个字的节点放入寄存器 A 中的编码，就从

```
                              LD1  I
LD1  I                        LDA  BASE(0:2)
LDA  L0,1     比如说，改变成    STA  *+1(0:2)        (8)
                              LDA  *,1
```

其中 BASE 含有 | L_0 | 0 | 0 | 0 |。这样的相对寻址明显地要比固定基地址寻址花费更长时间，尽管如果 MIX 有一个“间接寻址”的特性，它将只是略微慢一点(见习题 3)。

当每个可变大小的表是一个栈时，出现一个重要的特殊情况。这个时候，由于在任何时刻，只有每个栈的顶部元素是有关系的，因此我们几乎可以像以前一样有效地进行。假设我们有 n 个栈；如果对于第 i 个栈，BASE[i]和 TOP[i]是链接变量，而且每个节点都是一个字长，则上面的插入和删除算法变成

插入：TOP[i]←TOP[i] + 1；如果 TOP[i] > BASE[$i+1$]，则 OVERFLOW；
否则置 CONTENTS(TOP[i]) ← Y　　　　(9)

删除：如果 TOP[i] = BASE[i]，则 UNDERFLOW；否则，置
Y ← CONTENTS(TOP[i])，TOP[i] ← TOP[i] − 1　　　　(10)

这里 BASE[$i+1$]是第 $i+1$ 个栈的基地址。条件 TOP[i] = BASE[i]意味着栈 i 为空。

在(9)中，OVERFLOW 不再像它以前那样是一个危机了；我们可以“重新组装存储器”，从还未填满的表中腾出空间给已经溢出的表。进行存储重装的许多方法都各有特点；我们现在将详细地考虑其中的一部分，因为当对线性表进行顺序分配时，它们可能是十分重要的。我们将通过给出最简单的方法开始，然后将考虑别的。

假设有 n 个栈,而且值 BASE[i]和 TOP[i]要像(9)和(10)中那样加以操作。如果这些栈都共享由满足 $L_0 < L \leq L_\infty$ 的所有单元 L 组成的一个公共的存储区域。(这里 L_0 和 L_∞ 是确定可供使用的总字数的常数)。我们可以从所有栈都为空且

$$\text{BASE}[j] = \text{TOP}[j] = L_0, \qquad 对于\ 1 \leq j \leq n \tag{11}$$

开始。我们也置 BASE[$n+1$] = L_∞,使得对于 $i = n$,(9)还能正确地工作。

当相对于栈 i,OVERFLOW 出现时,存在三种可能性:

a)我们求出使 $i < k \leq n$ 和 TOP[k] < BASE[$k+1$]的最小的 k,如果有这样的 k 存在的话。现在上移一个节点:

置 CONTENTS(L+1)←CONTENTS(L),对于 TOP[k] ≥ L > BASE[$i+1$]

(这必须对于递减的而不是递增的 L 值进行以避免丢失信息。有可能 TOP[k] = BASE[$i+1$],在这种情况下,不需要任何移动。)最后,对于 $i < j \leq k$,置 BASE[j]←BASE[j]+1,和 TOP[j]←TOP[j]+1。

b)找不到 a)中那样的 k,但是我们可以找到使 $1 \leq k < i$ 和 TOP[k] < BASE[$k+1$]的最大的 k。现在我们下移一个节点:

置 CONTENTS(L−1)←CONTENTS(L),对于 BASE[$k+1$] < L ≤ TOP[i]

(这必须对于递增的 L 值进行。)然后对于 $k < j \leq i$,置 BASE[j]←BASE[j]−1 和 TOP[j]←TOP[j]−1。

c)对于所有的 $k \neq i$,我们有 TOP[k] = BASE[$k+1$]。于是显然我们不能为新的入栈项找到空间,因此必须放弃。

图 4 示出对于 $n = 4, L_0 = 0, L_\infty = 20$ 的情况,在逐次的动作

$$I_1^* \ I_1^* \ I_4 \ I_2^* \ D_1 \ I_3^* \ I_1 \ I_1^* \ I_2^* \ I_4 \ D_2 \ D_1 \tag{12}$$

之后存储器的格局。(这里 I_j 和 D_j 是指在栈 j 中的插入和删除,而星号是指 OVERFLOW 的出现,假定开始时对栈 1,2 和 3 都未分配空间。)

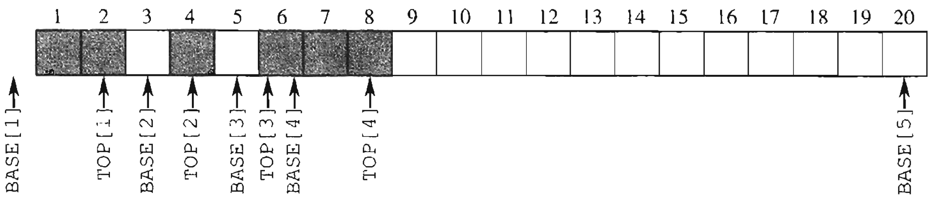


图 4 在若干次插入和删除之后存储器格局的例子

显然,如果我们明智地选择初始条件,而不是如同(11)所建议的那样,开始时就把所有空间分配给第 n 个栈,则使用这个方法出现的头一个栈的许多溢出都可以消去。例如,如果我们预料每个栈都有相同的大小,则可以以

$$\text{BASE}[j] = \text{TOP}[j] = \left\lfloor \left(\frac{j-1}{n}\right)(L_\infty - L_0) \right\rfloor + L_0, \ 对于\ 1 \leq j \leq n \tag{13}$$

开始。对于一个特定程序的操作经验可能会提议更好的启始值;然而,无论初始条

件设置得如何好,它最多能减少固定数目的溢出,而且只有在程序运行的早期阶段其效果才能察觉。(见习题 17。)

改进上述方法的另一个可能途径是在每次重装存储器时为多于一个新项提供空间。这个想法已被 J. Garwick 所利用。他建议,当出现溢出时,基于自上次存储重装以来每个栈的大小的变化,对存储器进行完全的重装。他的算法使用称做 OLDTOP[j]的一个另外的数组,$1 \leq j \leq n$,它保留在以前对存储器的分配之后 TOP[j]所拥有的值。开始时,和以前一样,以 OLDTOP[j] = TOP[j]来对这些表格设置值。算法如下:

算法 G(重新分配顺序表格)　假定按照(9),OVERFLOW 在栈 i 处出现。在实施算法 G 之后,我们将发现或者超出了存储容量,或者已对存储器重新进行了安排,使得可以执行动作 NODE(TOP[i])←Y。(注意,在算法 G 发生之前,在(9)中 TOP[i]已经增大了。)

G1. [初始化] 置 SUM←$L_\infty - L_0$,INC←0。然后对于 $1 \leq j \leq n$ 执行步骤 G2。(其效果将是使 SUM 等于剩下的存储空间的总量,INC 等于自上一次的分配以来表格大小增加的总量。)在完成了这一步之后,转到步骤 G3。

G2. [收集统计数字]置 SUM←SUM − (TOP[j] − BASE[j])。如果 TOP[j] > OLDTOP[j],则置 D[j]←TOP[j] − OLDTOP[j],以及 INC←INC + D[j];否则置 D[j]←0。

G3. [存储器满了吗?] 如果 SUM < 0,我们便不能进行。

G4. [计算分配因子] 置 α←0.1 × SUM/n,β←0.9 × SUM/INC。(这里 α 和 β 都是分数,不是整数,要对它们计算到合理的精度。下列步骤向个别表提供可用空间:将近 10%的当前可用存储单元将在 n 个表中平等地享用,其它的 90%将按照自前一次分配以来表格大小增加数量的比例进行划分。)

G5. [计算新的基地址] 置 NEWBASE[1]←BASE[1]和 σ←0;然后对于 $j = 2, 3, \cdots, n$,置 $\tau \leftarrow \sigma + \alpha +$ D[$j-1$]β,NEWBASE[j]←NEWBASE[$j-1$] + TOP[$j-1$] − BASE[$j-1$] + $\lfloor\tau\rfloor - \lfloor\sigma\rfloor$,且 $\sigma \leftarrow \tau$。

G6. [重装] 置 TOP[i]←TOP[i] − 1。(这反映第 i 个表的真实大小,使得不会在表的边界之外企图移动信息。)实施下面的算法 R,然后恢复 TOP[i]←TOP[i] + 1。最后对于 $1 \leq j \leq n$,置 OLDTOP[j]←TOP[j]。 ▎

或许这整个算法最有趣的部分是一般的重装过程,我们现在来描述它。由于重装时存储器的某些部分上移而其它部分下移,因此重装并非微不足道的事。显然重要的是移动时不去清除存储器中任何有用信息。

算法 R(重新放置顺序表格)　对于 $1 \leq j \leq n$,同上面指出的约定相一致,把由 BASE[j]和 TOP[j]所确定的信息移动到由 NEWBASE[j]所确定的新位置,而且

BASE[j]和 TOP[j]的值也被适当地调整。这个算法是基于以下容易验证的事实，即要被向下移动的数据不能同要被向上移动的数据重叠，也不能同应当保持在原处的任何数据重叠。

R1. [初始化] 置 $j \leftarrow 1$。

R2. [寻找移动的起点] (现在要下移的从 1 到 j 的所有表都被移动到所要求的位置。)逐步对 j 加 1 直到找到

a) NEWBASE[j] < BASE[j]：转到 R3；或者

b) $j > n$：转到 R4。

R3. [表下移] 置 $\delta \leftarrow$ BASE[j] − NEWBASE[j]。对于 L = BASE[j] + 1, BASE[j] + 2, …, TOP[j]，置 CONTENTS(L − δ) ← CONTENTS(L)。(BASE[j]有可能等于 TOP[j]，在此情况下不要求有任何动作。)置 BASE[j] ← NEWBASE[j]，TOP[j] ← TOP[j] − δ。转回步骤 R2。

R4. [寻找移动的起点] (现在要上移的从 j 到 n 的所有表都已被移动到所要求的位置。)逐步对 j 减 1 直到发现

a) NEWBASE[j] > BASE[j]：转到 R5；或者

b) $j = 1$：算法终止。

R5. [表上移] 置 $\delta \leftarrow$ NEWBASE[j] − BASE[j]。对于 L = TOP[j], TOP[j] − 1, …, BASE[j] + 1，置 CONTENTS(L + δ) ← CONTENTS(L)。(如同在步骤 R3 一样，在这里可能实际上不需要有动作)。置 BASE[j] ← NEWBASE[j]，TOP[j] ← TOP[j] + δ。转回到 R4。▌

注意栈 1 根本不需要移动。因此如果我们知道哪一个栈将是最大的，我们就先放置最大的栈。

在算法 G 和 R 中，我们已经有意地使对于 $1 \leq j \leq n$，有可能有

$$\text{OLDTOP}[j] \equiv \text{D}[j] \equiv \text{NEWBASE}[j+1]$$

即这三个表格可以共享存储单元，因为绝不会在冲突时需要它们的值。

我们已经对栈描述了这些重装算法，但显然它们也可被修改用于任何相对编址的表格，其中当前信息被包含在 BASE[j]和 TOP[j]之间。其它的指针(例如，FRONT[j]和 REAR[j])也可被附加到这些表上，使它们成为一个队列或双端队列。参见习题 8，它详细考虑了一个队列的情况。

对像上面所述的动态存储分配算法进行数学分析是极其困难的。以下的习题中有一些有趣的结果，尽管就一般特性而言，它们只是开始触及了一鳞半爪而已。

作为可以加以推导的理论的例子，假设我们考虑表格仅通过插入而增长的情况；而忽略掉删除和随后可以抵消其效果的插入。让我们进一步假定，每个表格都将以相同的速度被填入。这个状况可以通过想像 m 个插入操作的序列 $a_1, a_2, \cdots, a_m$ 来加以模拟，其中每个 a_i 是 1 和 n 之间的一个整数(表示在栈 a_i 顶部的一个插入)。例如，序列 1,1,2,2,1 表示对栈 1 的两个插入，接着是对于栈 2 的两个插入，

再接着是对栈 1 的另一个插入。我们可以认为 n^m 个可能的描述 $a_1, a_2, \cdots, a_m$ 的任何一个都是同等的。然后我们可以求，当整个表格被构造时，在重装操作期间，把一个字从一个单元移到另外一个单元所需要的平均移动次数。对于头一个算法，以对第 n 个栈给出的所有可用空间开始，习题 9 中对这个问题进行了分析。我们求得，所需要的移动操作的平均次数是

$$\frac{1}{2}\left(1-\frac{1}{n}\right)\binom{m}{2} \tag{14}$$

因此，如同我们预料的一样，移动的次数实际上同表格增长次数的平方成正比。如果个别的栈不是同等的，则结果也为真(参见习题 10)。

实际的教训似乎是，如果把相当多的项放进表格中，需要做很多的移动。这是为了把大量的顺序表格紧凑地组装在一起所必须支付的代价。分析算法 G 的平均特性的理论还没有建立起来；因而在这样的环境中不可能有任何简单的模型能够描述现实生活中的表格的特征。然而，习题 18 提供了一个最坏情况的保证，即如果存储器不太满，则运行时间将不至于太坏。

经验证明，当存储器仅有一半负载(即当可用空间等于总空间的一半)时，对于算法 G 我们需要非常少的对表格的重新安排。重要之点或许是，在一半满的情况下算法很好地动作，而在几乎满的情况下，它至少提供正确的答案。

但且让我们更仔细地考虑几乎满的情况。当表格近乎满时，算法 R 花费相当长时间来执行它的工作。而且使事情更糟的是，在存储器空间快被用完之前 OVERFLOW 发生得更为经常。有非常少的程序，它们将很接近于把存储器填满但却不会很快完全溢出；而那些使存储器溢出的程序在存储器被用完之前大概将在算法 G 和算法 R 中花费大量的时间。不幸的是，未调试过的程序将经常使存储器溢出。为避免浪费这些时间，一个可能的建议将是，如果 SUM 比 $S_{\min}$ 小，则在步骤 G3 停止算法 G，其中 $S_{\min}$ 由程序员选择以防止过多的重装。当有许多可变大小的顺序表格时，在超出存储容量之前，我们不应期望能利用百分之百的存储空间。

D. S. Wise 和 D. C. Watson 对算法 G 做了进一步的研究，见 *BIT* **16** (1976), 442 ~ 450。也可参见 A. S. Fraenkel, *Inf. Proc. Letters* **8** (1979), 9 ~ 10，他建议使用彼此相向增长的栈偶进行工作。

习　题

▶ 1. [*15*] 在由(6a)和(7a)给出的队列操作中，如果不出现 OVERFLOW，在队列中一次可以放进多少项？

▶ 2. [*22*] 推广(6a)和 7(a)的方法使得它将适用于有少于 M 个元素的任何双端队列。换言之，给出对于其它两个操作，“从后端删除”和“在前端插入”的说明。

3. [*21*] 假设把 MIX 扩展如下：每个指令的 I 字段有 $8I_1 + I_2$ 的形式，其中 $0 \le I_1 < 8, 0 \le$

$I_2<8$。在汇编语言中,人们写“OP ADDRESS,I_1:I_2”,或者(像现在这样)如果 $I_1=0$,“OP ADDRESS,I_2”。意思是首先对 ADDRESS 实现“地址修改”I_1,然后对得到的地址实现“地址修改”I_2,最后对于新地址执行 OP。地址修改定义如下:

0:M = A

1:M = A + rI1

2:M = A + rI2

…

6:M = A + rI6

7:M = 在单元 A 找到的由“ADDRESS,I_1:I_2”所定义的地址。不允许在单元 A 中出现 $I_1=I_2=7$ 的情况。(习题 5 中讨论了后面这一个限制的原因。)

这里 A 表示操作之前的地址,而 M 表示地址修改后得到的地址。在所有情况下如果 M 的值装不进两个字节和一个符号中,结果就是无定义的。对于所执行的每个“间接寻址”的(即修改 7)操作,执行时间增加 1 个单位时间。

作为一个非平凡的例子,假设单元 1000 包含“NOP 1000,1:7”;单元 1001 包含“NOP 1000,2”;且变址寄存器 1 和 2 分别包含 1 和 2。则命令“LDA 1000,7:2”等价于“LDA 1004”,因为

1000,7:2 = (1000,1:7),2 = (1001,7),2 = (1000,2),2 = 1002,2 = 1004

a)使用这个间接寻址的特征(如果必要的话),说明如何简化(8)的右边代码,使得每次访问这个表格可节省两条指令。你的程序要比(8)的程序快多少?

b)假设有若干个表格,它们的基地址存储在单元 BASE + 1,BASE + 2,BASE + 3,…;如何使用间接寻址特性把第 J 个表格的第 I 个单元在一条指令中放进寄存器 A? 假定 I 在 rI1 中而 J 在 rI2 中。

c)假定单元 X 的(3:3)字段为零,问指令“ENT4 X,7”的效果是什么?

4.[*25*] 假定 MIX 已像在习题 3 中那样被扩充,试说明对下列每一个动作如何给出一单条指令(加上辅助常数):

a)由于间接寻址不终止而无穷地循环。

b)把 LINK (LINK (x))的值放进寄存器 A 中,其中链接变址 x 的值被存储在其符号地址为 X 的单元的(0:2)字段中,LINK(x)的值被存储在单元 x 的(0:2)字段中,等等,假定在这些单元的(3:3)字段为零。

c)在类似于 b)的那些假定下,把 LINK (LINK (LINK (x)))的值放进寄存器 A 中。

d)把单元 rI1 + rI2 + rI3 + rI4 + rI5 + rI6 的内容放进寄存器 A 中。

e)把 rI6 当前的值放大到四倍。

▶ 5.[*35*] 习题 3 中建议的对 MIX 的扩充有一个不幸的限制,即在一个间接寻址单元中不允许有“7:7”。

a)试给出一个例子指出,如果没有这个限制,大概对于 MIX 的硬件来说需要有能力维护一个长的三位项的内部栈。(即使是对于像 MIX 这样一台虚构的计算机,这都将是极其昂贵的硬件。)

b)说明为什么在现有的限制下,这样的栈是不需要的;换句话说,试设计一个算法,通过这个算法,不需要太多额外的寄存器容量,计算机硬件就可实现所要求的地址修改。

c)试给出一个比习题 3 中对 7:7 的使用要稍微宽松些的限制,它可以减轻习题 4c)的困难程度,而且在计算机的硬件上可以经济地实现。

6. [*10*] 以图 4 所示的存储格局开始，确定下列操作序列哪一个引起上溢或下溢？

(a)I_1；(b)I_2；(c)I_3；(d)$I_4I_4I_4I_4I_4$；(e)$D_2D_2I_2I_2I_2$。

7. [*12*] 算法 G 的步骤 G4 指出由量 INC 来除的一个除法。在算法的该点处 INC 能成为零吗？

▶ 8. [*26*] 对于有一个或多个表是像在(6a)和(7a)中那样被循环地处理的队列的情况，怎样修改(9)，(10)以及重装算法？

▶ 9. [*M27*] 使用接近正文末尾处描述的数学模型，证明等式(14)是预期的移动次数。(注意序列 1,1,4,2,3,1,2,4,2,1 确定 0+0+0+1+1+3+2+0+3+6=16 个移动。)

10. [*M28*] 试修改习题 9 的数学模型，使得预期某些表格要比其它的表格更大些：令 p_k 是对于 $1\leq j\leq m, 1\leq k\leq n, a_j=k$ 的概率。因此 $p_1+p_2+\cdots+p_n=1$；上一道题考虑了对于所有 k，$p_k=1/n$ 的特殊情况。试对于更一般的情况，如同在等式(14)那样，确定预期的移动次数。有可能重新安排 n 个表的相对顺序使得把预期要长些的表放置在预期要短些的表的右边(或左边)；基于 $p_1,p_2,\cdots,p_n$，什么样的 n 个表的相对次序将使预期的移动次数最少？

11. [*M30*] 试推广习题 9 的论证使得在任何栈内的头 t 个插入不会引起移动，而随继的插入不受影响。因此如果 $t=2$，习题 9 中的序列确定 0+0+0+0+0+3+0+0+3+6=12 次移动。在这个假定之下平均的总共移动次数是多少？[这是对每个栈以 t 个可用空间开始时，算法特性的近似。]

12. [*M28*] 在内存中有两个表格共存并且彼此相向增长，而不是使它们保持在分开的独立的受限区域中，其优点可以(在某种程度上)定量地估计如下：以 $n=2$ 使用习题 9 的模型；对于 2^m 个同样可能的序列 $a_1,a_2,\cdots,a_m$ 的每一个，设有 k_1 个 1 和 k_2 个 2。(这里 k_1 和 k_2 是在内存满了之后，两个表格各自的大小。当两个表格相邻时，我们能够以 $m=k_1+k_2$ 个单元来运行算法，而不必以分开的表格，用 $2\max(k_1,k_2)$ 个单元获得相同的效果。)

$\max(k_1,k_2)$ 的平均值是多少？

13. [*HM42*] 如果通过允许随机删除以及随机插入而在表格中引入更大的起伏波动，则在习题 12 中所研究的 $\max(k_1,k_2)$ 的值将会更大。假设我们改变这个模型使得序列的值 a_j 被解释作一个删除而不是一个插入的概率为 p；这一过程继续直到 k_1+k_2(使用中表格单元的总数)等于 m 为止。从一个空表的删除不起作用。

例如，如果 $m=4$，可以证明，当过程停止时，我们得到下列的概率分布：

$(k_1,k_2)=$　(0,4)　(1,3)　(2,2)　(3,1)　(4,0)

其概率为

$$\frac{1}{16-12p+4p^2},\quad \frac{1}{4},\quad \frac{6-6p+2p^2}{16-12p+4p^2},\quad \frac{1}{4},\quad \frac{1}{16-12p+4p^2}$$

因此当 p 增加时，k_1 和 k_2 之间的差别趋向于增大。不难证明，当 p 趋于 1 时在这个极限中，k_1 的分布实际上变成为一致的，而 $\max(k_1,k_2)$ 极限的预期值恰好是 $\frac{3m}{4}+\frac{1}{4m}[m\text{ 奇}]$。这一特征十分不同于在上题中(当 $p=0$ 时)的特征；然而，它并非极有意义的，因为当 p 趋于 1 时，为终止这个过程所花费的时间迅速地趋于无穷大。本题中提出的问题是考察 $\max(k_1,k_2)$ 对于 p 和 m 的依赖性，并且确定当 m 趋于无穷大时对于固定的 p(比如 $p=\frac{1}{3}$)的渐近公式。$p=\frac{1}{2}$ 的情况是特别有趣的。

14. [*HM43*] 通过证明，当 n 固定和 m 趋于无穷大时，量

$$\frac{m!}{n^m}\sum_{\substack{k_1+k_2+\cdots+k_n=m\\k_1,k_2,\cdots,k_n\geq 0}}\frac{\max(k_1,k_2,\cdots,k_n)}{k_1!\,k_2!\cdots k_n!}$$

有渐近形式 $m/n+c_n\sqrt{m}+O(1)$,把习题 12 的结果推广到任意的 $n\geq 2$ 上。试确定常数 c_2,c_3,c_4 和 c_5。

15. [*40*] 使用蒙特卡罗方法,模拟在插入和删除变化的分布之下算法 G 的特征。关于算法 G 的效率你的实验意味着什么?试将其性能与较早前给出的每次上移和下移一个节点的算法作一比较。

16. [*20*] 正文说明如何放置两个栈使得它们彼此相向增长,由此高效地利用公共存储区域。两个队列,或者一个栈与一个队列,能否以同样的效率利用公共存储区域?

17. [*30*] 如果 σ 是如同(12)那样的任何插入和删除的序列,令 $s_0(\sigma)$是当把图 4 的简单方法以初始条件(11)应用于 σ 时出现的栈溢出的个数,且令 $s_1(\sigma)$是相对于(13)那样另一个初始条件对应的溢出的个数。证明 $s_0(\sigma)\leq s_1(\sigma)+L_\infty-L_0$。

▶ 18. [*M30*] 证明通过算法 G 和 R,对于任何 m 个插入和/或删除序列的总共运行时间是 $O(m+n\sum_{k=1}^{m}\alpha_k/(1-\alpha_k))$,其中 α_k 是在第 k 个操作之前在最近的重装上占用的存储的比例;在头一次重装之前 $\alpha_k=0$。(比如,如果存储器不会达到 90% 满,则不管总的存储器的大小是多少,在平摊的意义下,每个操作至多花费 $O(n)$时间单位。)假定 $L_\infty-L_0\geq n^2$。

▶ 19. [*16*](从 0 开始的下标) 有经验的程序员知道,一般以 X[0],X[1],…,X[$n-1$]来表示一个线性表的元素,而不是使用更传统的记号 X[1],X[2],…,X[n]是明智的。例如在(1)中的基地址 L_0 指向数组的最小单元。

试修改对于栈和队列的插入和删除方法(2a),(3a),(6a)和(7a),使得它们同这个约定相一致。换句话说,把它们改变成使表元素将在数组 X[0],X[1],…,X[M-1],而不是 X[1],X[2],…,X[M]中出现。

2.2.3 链接分配

我们可以利用一个灵活得多的方案代替在顺序内存单元中保存一个线性表,其中每个节点包含对于这个表的下一节点的链接。

顺序分配

地址	内容
L_0+c:	项 1
L_0+2c:	项 2
L_0+3c:	项 3
L_0+4c:	项 4
L_0+5c:	项 5

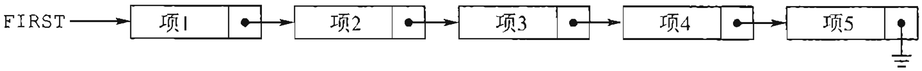
链接分配

地址	内容	
A:	项 1	B
B:	项 2	C
C:	项 3	D
D:	项 4	E
E:	项 5	Λ

这里 A,B,C,D 和 E 是在内存中的任意单元,而 Λ 是空链接(参见 2.1 节)。在顺序分配情况下使用这个表格的程序将有一个附加的变量或常数,其值表示这个表格的长度为 5 个项,或者不然的话这一信息将通过在项 5 之内的一个标记代码或在下一个单元中

指明。用于进行链接分配的程序将有指向 A 的一个链接变量或链接常数;这个表的所有其它项都可从地址 A 找到。

由 2.1 节可以知到,链接通常简单地以箭头表示,因为所占有的真正内存单元通常是无关紧要的。因此上面的链接表格可以表示如下:



(1)

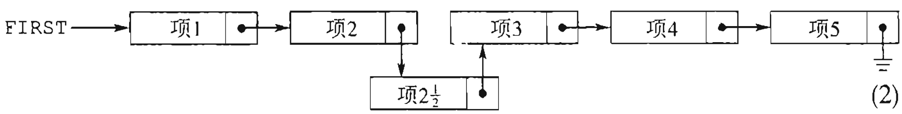
这里 FIRST 是指向表的头一个节点的链接变量。

我们可以对这两种基本形式的存储作若干明显的比较:

1)链接分配占用额外的存储空间保存链接信息。在某些情况下这可能是一个支配性因素。然而,我们经常发现,在一个节点中的信息无论如何并不占满整个字,因此已经留有用作一个链接字段的空间。而且在许多应用中,有可能把若干个项组合到一个节点上,使得对于若干个信息项仅有一个链接(见习题 2.5-2)。而甚至更为重要的是,通过链接存储方法,在存储容量方面,通常还有一个隐含的好处,因为表格可以重叠、共享公共部分;而且在许多情况下,顺序分配将没有链接分配那样有效率,除非有一个相当大量的额外的内存单元保持空闲。例如,在上一小节末尾的讨论说明了为什么当内存被稠密地装入时,所描述的系统必然低效。

2)从一个链接表内很容易删除一个项。例如,为了删除项 3,我们只须改变同项 2 相关联的链接。但对于顺序分配,这样一个删除一般都意味着要把一大部分表上移到不同的单元中去。

3)当使用链接方案时,把一个项插入到一个表的中间是很容易的。例如,为了把一个项 $2\frac{1}{2}$ 插入到(1)中我们仅须改变两个链接:



(2)

比较起来这个操作在一个长的顺序表格中是极其浪费时间的。

4)在顺序情况下对表的随机部分的访问要快得多。为了获得对表中第 k 项的访问,当 k 是一个变量时,在顺序情况下花费一个固定的时间即可,但是在链接的情况下我们需要 k 次迭代向下前进到正确的位置。因此链接存储的有用性是基于以下事实来论断的,即在绝大多数的应用中我们要顺序地通过表,而不是随机地;如果需要表中间的项或者底部的项,我们可以试图保持一个附加的链接变量或者链接变量表以指向适当位置。

5)链接方案使得较易于把两个表联合在一起,或者把一个表分开成为独立地增长的两个表。

6)链接方案使它本身立即具有比简单的线性表更复杂的结构。我们可以有个数变化和大小变化的表;表的任何节点可以是另一个表的起始点;节点可以以对应于不同表的若干种顺序同时地链接在一起;等等。

7)像顺序地处置一个表这样的简单操作,在许多计算机上对于顺序表来说是稍微更快的。对于 MIX 来说,比较是在“INC1 c”和“LD1 0,1(LINK)”之间进行的,区别只不过一个周期,但是许多机器并不具有从一个变址单元装入一个变址寄存器的能力。如果一个链接表的元素属于一个海量存储器中的不同页,则存储器访问可能要更久。

因此我们看到,链接技术,它使我们从由计算机存储器的连续性引起的任何限制中解放出来,在某些操作中确实给了我们大量的效率,但同时在某些情况下使我们损失某些能力。通常在一个给定的情况下,哪一种分配技术最适当是明显的,因此两种方法通常用于相同程序的不同表中。

在下面的一些例子中,为方便起见,我们假定一个节点有一个字并且它被分成两个字段 INFO 和 LINK:

INFO	LINK

(3)

使用链接分配一般意味着,当我们希望把某个新近建立的信息插入到一个表时,存在有为一个新节点寻找可用空闲空间的某种机制。这通常是通过叫做*可用空间表*的一个特殊表完成的。我们将称之为 AVAIL 表(或者 AVAIL 栈,因为通常它是以后进先出方式而被处理的)。当前未被使用的所有节点的集合被链接在一个表中,就如同任何其它别的表一样。因此,如果要把链接变量 X 置成一个新节点的地址,以及保留该节点以供将来使用,我们可以如下进行:

$$X \leftarrow \text{AVAIL}, \text{AVAIL} \leftarrow \text{LINK}(\text{AVAIL}) \tag{4}$$

这有效地删除 AVAIL 栈的顶部和使 X 指向刚刚删除的节点。*操作(4)如此经常地出现,因而我们对它有一个特殊的记号*:“X⇐AVAIL”将表示把 X 置成指向一个新节点。

当删除一个节点而且不再需要它时,可逆转进程(4):

$$\text{LINK}(X) \leftarrow \text{AVAIL}, \ \text{AVAIL} \leftarrow X \tag{5}$$

这个操作把由 X 编址的节点送回到原始材料的表中;我们以“AVAIL⇐X”来标记(5)。

在关于 AVAIL 栈的讨论当中我们已省略了若干重要的东西。我们没有谈及在一个程序开始时如何设置这个栈;显然这可以通过下列步骤来完成:(a)把准备用作链接存储的所有节点链接在一起,(b)把 AVAIL 置成这些节点的头一个的地址,以及(c)把最后的节点链到 Λ。所有可被分配节点的集合叫做*存储池*。

在我们的讨论中一个更为重要的省略是对溢出的测试。我们忽略了检查在(4)中是否所有可用的存储空间都被使用了。操作 X⇐AVAIL 实际上应被定义如下:

如果 AVAIL = Λ,则 OVERFLOW;

否则 X ← AVAIL, AVAIL ← LINK(AVAIL) (6)

对溢出的可能性总须加以考虑。这里 OVERFLOW 一般意味着我们遗憾地终止程序;或者我们可以转到试图找到更多可用空间的一个“废料(一作无用信息)收集”程序。在 2.3.5 小节将讨论废料收集。

还有另外一个处理 AVAIL 栈的重要技术:我们通常事先并不知道对于存储池应当使用多少存储空间。可能有一个大小可变的顺序表格,要同链接表格共存于存储器中;在这样一种情况下我们不希望链接存储区域比绝对需要的用更多的空间。所以假设我

们希望以递升的单元来放置链接存储区域,该区域以 L_0 开始并且它绝不扩展到超出变量 SEQMIN(它表示顺序表格当前的下限)的值,那么使用一个新变量 POOLMAX 我们可以如下进行:

a)开始时置 AVAIL←Λ 以及 POOLMAX←L_0。

b)操作 X⇐AVAIL 变成如下:

"如果 AVAIL≠Λ,则 X←AVAIL,AVAIL←LINK(AVAIL);

否则置 X←POOLMAX 且 POOLMAX←X + c,其中 c 是节点大小;如果 POOLMAX > SEQMIN,则 OVERFLOW 现在发生。"　(7)

c)当程序的其它部分试图减小 SEQMIN 的值时,如果 SEQMIN < POOLMAX 它们应当发出 OVERFLOW 的警报。

d)由(5),操作 AVAIL⇐X 不变。

这一思想实际上比以前的方法多不出多少东西,只是以一个特殊的恢复过程代替(6)中 OVERFLOW 的情况。其净效果是保持存储池尽可能小。许多人喜欢使用这一思想,即使当所有表占据存储池区域(使得 SEQMIN 是常数)时,因为它避免了开始时把所有变量单元链接在一起的相当浪费时间的操作,而且它容易排错。当然,我们可以把顺序表放在下部和把池放在顶部,由 POOLMIN 和 SEQMAX 来代替 POOLMAX 和 SEQMIN。

因此,以这样一种方式,即自由节点可有效地找到且过后被返回,维护可用节点的池就十分容易了。这些方法给了我们用于链接表格的原始材料来源。我们的讨论是基于隐含的假设,即所有节点都有一个固定的大小 c;当出现不同大小的节点时导致的结果非常重要,但我们将把其讨论推迟到 2.5 节。现在我们将在涉及栈和队列的特殊情况下考虑一些最普遍的表操作。

最简单的链接表类型是栈。图 5 示出一个典型的栈,带有指向栈顶的一个指针 T。当这个栈为空时,这个指针将有值 Λ。

使用一个辅助的指针变量 P,如何把新信息 Y 插入("压入")这样一个栈的顶部是清楚的。

P⇐AVAIL, INFO (P) ← Y, LINK(P) ← T, T ← P　(8)

反过来,要把 Y 置成等于栈顶部处的信息和"弹出"栈:

如果 T = Λ,则 UNDERFLOW;

否则,置 P←T,T←LINK (P),Y⇐INFO (P),

AVAIL⇐P　(9)

应该把这些操作同 2.2.2 小节中的顺序分配栈的类似机制(2a)和(3a)相比较。读者应仔细研究(8)和(9),因为它们是极其重要的操作。

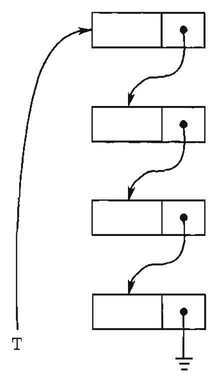


图 5　一个链接的栈

在考察队列的情况以前,让我们看看在 MIX 的程序中,如何能方便地表达栈操作。一个插入程序,且 P≡rI1,可写作如下:

```
INFO    EQU    0:3    定义 INFO 字段
LINK    EQU    4:5    定义 LINK 字段
```

```
LD1     AVAIL           P←AVAIL               }
J1Z     OVERFLOW        AVAIL = Λ 吗?         }  P⇐AVAIL
LDA     0,1(LINK)                             }
STA     AVAIL           AVAIL←LINK(P)         }              (10)
LDA     Y
STA     0,1(INFO)       INFO(P)←Y
LDA     T
STA     0,1(LINK)       LINK(P)←T
ST1     T               T←P              ▌
```

同对一个顺序表格的相当操作的 12 个时间单位相对照,这个程序花费 17 个时间单位(尽管顺序时在很多情况中 OVERFLOW 要花费长得多的时间)。在这个程序中,如同在本章后面的其它程序中一样,OVERFLOW 或者表示一个结束的程序,或者表示寻找更多空间并返回到单元 rJ – 2 的子程序。

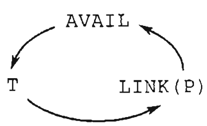
用于进行删除的程序也同样简单:

```
LD1     T               P←T
J1Z     UNDERFLOW       T = Λ 吗?
LDA     0,1(LINK)
STA     T               T←LINK(P)                            (11)
LDA     0,1(INFO)
STA     Y               Y←INFO(P)
LDA     AVAIL                           }
STA     0,1(LINK)       LINK(P)←AVAIL   } AVAIL⇐P
ST1     AVAIL           AVAIL←P         }                    ▌
```

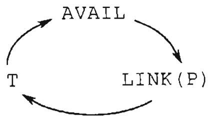
观察这样一点是有趣的,即这些操作的每一个都涉及三个链接的一个循环排列。例如,在插入操作中,令 P 是插入之前 AVAIL 的值;如果 P≠Λ,我们发现,在这个操作之后,

AVAIL 的值已经变成 LINK(P)以前的值,
LINK(P)的值已经变成 T 以前的值,以及
T 的值已经变成 AVAIL 以前的值

因此插入过程(除了设置 INFO (P)←Y 外)是循环排列



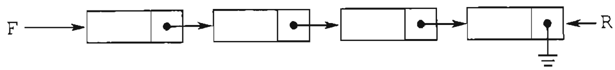
类似地,在删除的情况下,其中 P 有在此操作前 T 的值,而且假定 P≠Λ,我们有 Y←INFO(P),且



排列是循环的事实并非真正有关系的课题，因为移动每一个元素的任何三个元素的排列都是循环的。重要之点倒是在这些操作中恰有三个链接被排列。

(8)和(9)的插入和删除算法是对栈描述的，但它们同样适用于在任何线性表中更一般得多的插入和删除。例如，插入是在由链接变量 T 指出的节点的紧前边实施的。上面在(2)中的项 $2\frac{1}{2}$ 的插入将通过使用(8)以及 T = LINK(LINK (FIRST))来完成。

链接分配以一个特别方便的方式适用于队列。在这种情况下容易看到，链接应当从队列的前端向后端进行，使得当从前端删去一个节点时，新的前端节点就直接地被确定。我们将利用指针 F 和 R 来指向前端和后端：

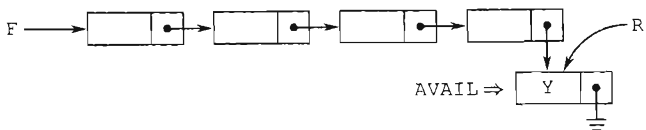


(12)

除 R 外，这个框图抽象地说等同于前面的图 5。

每当设计了一个表的布局时，重要的是小心地确定所有的条件，特别是对于表为空的情况。同链接分配有关的最普遍的程序设计错误之一是没有适当地处理空表；另一个常见错误是当一个结构被操作时忘记改变某些链接。为了避免头一种类型的错误，我们应当仔细考察“边界条件”。为了避免造成第二种类型的错误，画出“之前和之后”的框图，并对它们加以比较是有帮助的，以便看出哪一些链接必须改动。

让我们通过把上述原则应用于队列的情况来说明上一段的论述。首先考虑插入操作：如果(12)是插入前的情况，则在队列末端作了插入之后的图形应当是

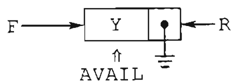


(13)

(这里所用记号意味着从 AVAIL 表已经得到一个新节点。)比较(12)和(13)，我们就看出当把信息 Y 插到队列后部时应如何进行：

$$P \Leftarrow \text{AVAIL},\ \text{INFO}(P) \leftarrow Y,\ \text{LINK}(P) \leftarrow \Lambda,\ \text{LINK}(R) \leftarrow P,\ R \leftarrow P \qquad (14)$$

现在让我们考虑当队列为空时的“边界”状况：在这种情况下，插入之前的状况尚有待确定，而“之后”的状况是



(15)

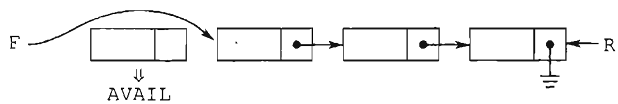
操作(14)也适用于这个情况是合意的，即使插入到一个空队列意味着我们必须改变 F 和 R 两者，而不仅仅是 R 也好。我们发现当队列为空时，假定 F≡LINK(LOC (F))，如果 R = LOC(F)，则(14)将有效；如果要使这个想法有效，变量 F 的值必须被存储在它的单元的 LINK 字段中。为了使得对一个空队列的测试尽可能地有效，在这种情况下我们

将令 F = Λ。因此我们的政策是：

一个空队列由 F = Λ *和* R = LOC(F)*来表示*

在这些情况下，如果应用操作(14)，我们便得到(15)。

以类似的方式可以导出对于队列的删除操作。如果(12)是删除之前的状况，则删除之后的状况是



(16)

对于边界条件，我们必须确保当在这个操作前后队列为空时删除操作有效。这些考虑使我们得出进行队列删除操作的下列一般方式：

如果 F = Λ，则 UNDERFLOW；
　　否则置 P←F，F←LINK(P)，Y←INFO(P)，AVAIL⇐P，
　　　　而且如果 F = Λ，则置 R←LOC(F)　　(17)

注意当队列变空时，R 必须改变；这正是我们时刻应加以考察的"边界条件"的类型。

这些建议不是以一个线性链接方式表示队列的惟一方式；习题 30 叙述了一个稍微更自然的选择，而且在本章稍后我们将给出其它方法。其实，上述这些操作中没有任何一个被规定为做某件事的惟一方法；它们仅仅被当做对链接进行操作的基本手段的例子。对于这样一些技术，以前只有较少经验的读者在往下阅读之前重新阅读本小节直到这里为止，将会发现有所帮助。

迄今为止在本章中我们已经讨论了如何对表格实施某些操作，但是如果不展示实际的程序和其中有用的技术，那我们的讨论总是"抽象的"。对于一个问题，在人们看到足够多的事例以引发他们的兴趣之前，一般不会有对该问题进行抽象研究的动机。迄今所讨论的操作——通过插入和删除来对可变大小的信息表的操作，以及作为栈或队列的表格的使用——有着如此广泛的应用，因此希望读者已经足够经常地碰到它们，这样就会认同我们所说的重要性。但在我们开始研究本章所讨论技术的一系列重要的实例时，我们将离开这个抽象的领域。

我们的头一个例子是所谓*拓扑排序*的问题，它是在同网络问题、同所谓的 PERT 图表甚至同语言学相关联中需要的一个重要过程；事实上，只要我们有一个涉及*偏序*的问题，它就有潜在的用途。一个集合 S 的偏序是在 S 的对象之间的一个关系，我们可以用符号"$\preceq$"来表示它，对于 S 中任何对象 x，y 和 z(不必是不相同的)，它满足以下性质：

i)如果 $x \preceq y$ 且 $y \preceq z$，则 $x \preceq z$。(传递性。)

ii)如果 $x \preceq y$ 且 $y \preceq x$，则 $x = y$。(反对称性。)

iii)$x \preceq x$。(反身性。)

记号 $x \preceq y$ 可以读作"x 居前于或等于 y"。如果 $x \preceq y$ 且 $x \neq y$，则我们写 $x \prec y$ 并说"x 居前于 y"。从 i)，ii)和 iii)容易看出，我们总有

i′)如果 $x \prec y$，且 $y \prec z$，则 $x \prec z$。(传递性。)

ii′)如果 $x \prec y$,则 $y \not\prec x$。(非对称性。)

iii′)$x \not\prec x$。(非反身性。)

由 $y \not\prec x$ 所表示的关系意味着“y 不居前于 x”。如果以满足性质 i′,ii′)和 iii′)的关系 $\prec$ 开始,则可以把上述过程颠倒过来并且定义若 $x \prec y$ 或 $x = y$,则 $x \preceq y$;那么性质 i),ii)和 iii)是正确的。因此我们可以认为性质 i),ii)和 iii)或者性质 i′),ii′)和 iii′)可作为偏序的定义。注意性质 ii′)实际上是 i′)和 iii′)的结果,尽管 ii)不是从 i)和 iii)得出的。

在日常生活中,如同在数学中一样,十分经常地出现偏序。作为来自于数学的例子,我们可以提及实数 x 和 y 之间的关系 $x \leqslant y$;对象的集合之间的关系 $x \subseteq y$;正整数之间的关系 $x \setminus y$(x 整除 y)。在 PERT 网络的情况下,S 是必须完成的工作的集合,而关系“$x \prec y$”意味着“x 必须在 y 之前完工”。

我们将自然地假定 S 是一个有限集合,因为我们要在一台计算机内对 S 进行处理。在一个有限集合上的偏序总可以通过绘制像图 6 那样的框图来说明,其中小框表示对象,而这些小框之间的箭头表示关系;$x \prec y$ 表示从标号为 x 的框到标号为 y 的框之间存在沿着箭头方向的一条通路。偏序的性质 ii)意味着在这个框图中没有封闭的循环(即没有在它们本身上封闭的通路)。如果在图 6 中从 4 到 1 画一个箭头,那我们将不再有一个偏序了。

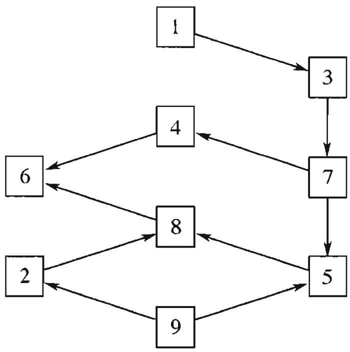


图 6　一个偏序

拓扑排序问题是把偏序嵌入到一个线性序中,即把对象安排成一个线性序列 $a_1 a_2 \cdots a_n$ 使得只要 $a_j \prec a_k$,那我们就有 $j < k$。从图形上说,这意味着方框要被重新安排成一条线,使得所有的箭头都指向右(见图 7)。这样的一种重新安排是否在每种情况下都是可能的,并不是显然的,尽管如果存在有任何循环,则这样一种重新安排肯定不可能完成。因此我们将要介绍的算法是有趣的,不仅是因为它实现一个有用的操作,而且因为它证明对于每个偏序这个操作都是可能的。

作为拓扑排序的一个例子,想像包含有技术术语定义的一个很大的词汇表,如果词 w_1 的定义直接或间接地依赖于词 w_2 的定义,我们可以写 $w_2 \prec w_1$。只要是没有“循环”定义,则这个关系是一个偏序。在这种情况下,拓扑排序的问题是在词汇表中找出安排词汇的一种方法,使得在一个术语被定义之前不会被使用。在编写程序来处理某个汇

编和编译程序语言中的说明当中,也出现类似的问题。它们也出现在编写描述一种计算机语言的用户手册或者在编写关于信息结构的教材当中。

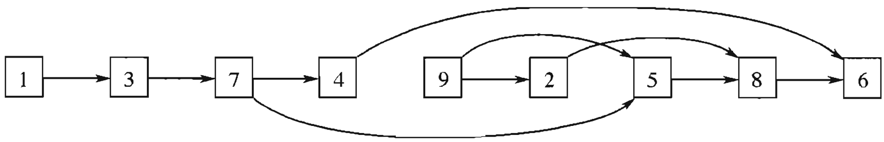


图 7 在拓扑排序之后图 6 的有序关系

进行拓扑排序有一个非常简单的方法:我们以在这个排序中不被任何其它对象居前的一个对象开始。这个对象可以首先放置在输出中。现在我们从集合 S 中删去这个对象;得到的集合再次是一个偏序集。因此这个过程可以被重复直到整个集合都已被排序为止。例如,在图 6 中我们可以通过删去 1 或 9 开始;在删去 1 之后,可以取 3,等等。这个算法可能失灵的惟一情况将是,如果有一个非空的偏序集,其中每个元素都被另一个元素所居前;因为在这样一种情况下,这个算法将发现无计可施。但是如果每个元素都被另一个元素所居前,我们就可以构造一个任意长的序列 $b_1,b_2,b_3,\cdots$,其中 $b_{j+1}<b_j$。由于 S 是有限的,我们必定有对于某个 $j<k$,$b_j=b_k$;但这意味着 $b_k\leq b_{j+1}$,同 ii)相矛盾。

为了通过计算机高效地实现这个过程,我们需要准备好实施上面所描述的动作,即放置不被任何其它对象所居前的对象,并从集合中删去它们。我们的实现也受到所要求的输入和输出特征的影响。最一般的程序将接受对象的字母名称,并允许对庞大的对象集合进行排序——这比一次可装入到计算机内存中的数量要多。然而这样的复杂化将模糊我们在这里试图澄清的要点;使用第 6 章的方法能有效地处理字母数据,而对大型网络的处理留给读者作为一个有趣的设计项目。

因此我们将假定,要加以排序的对象将以任何顺序编码成从 1 到 n。程序的输入将在带设备 1 上:每个带记录包含 50 对的数,其中数对(j,k)表示对象 j 居前于对象 k,然而头一个数对是$(0,n)$,其中 n 是对象的数目。数对(0,0)终止输入。我们将假定 n 加上关系对的数目将从容地存入内存中,而且我们还将假定没有必要检查输入的正确性。输出是排序之后对象的号码,后面跟着数字 0,在带设备 2 上。

作为输入的一个例子,我们可以有下列关系:

$$9<2,\ 3<7,\ 7<5,\ 5<8,\ 8<6,\ 4<6,\ 1<3,\ 7<4,\ 9<5,\ 2<8 \tag{18}$$

没有必要给出比表征所要求的偏序更多的关系来。因此,像 $9<8$ 这样的附加的关系(它可以由 $9<5$ 和 $5<8$ 导出)可以无害地从输入中删去或添加进来。一般地说,只需要给出与图 6 那样的框图上的箭头对应的关系。

以下的算法使用一个顺序表格 X[1],X[2],…,X[n],而且每个节点 X[k]都有下列形式:

+	0	COUNT[k]	TOP[k]

这里 COUNT[k]是对象 k 的直接前驱的数目(即在输入中已经出现的关系 $j<k$ 的数目),而且 TOP[k]是对对象 k 的直接后继的表开始处的一个链接。这后一个表包含如下格式下的项目:

+	0	SUC	NEXT

其中 SUC 是 k 的一个直接后继,而 NEXT 是表的下一项。作为这些约定的一个例子,图 8 示意了对应于输入(18)的内存内容。

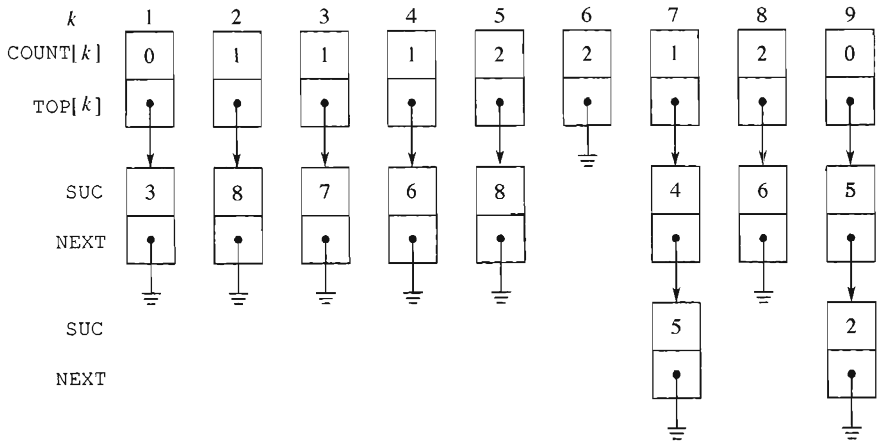


图 8　对应于关系(18)的图 6 的计算机表示

使用这个内存布局,不难把算法设计出来。我们要输出 COUNT 字段为零的节点,然后把这些节点的所有后继的 COUNT 字段都减 1。这个技巧是避免对 COUNT 字段为零的节点再去做任何的"查找",而这可以通过维持一个含有这些节点的队列来完成。对于这个队列的链接被保存在 COUNT 字段中,现在它可以为前边的目的服务。为了清楚起见,在下列的算法中,当 COUNT[k]字段不再被用来保存计数时,我们使用 QLINK[k]这个记号来表示它。

算法 T(拓扑排序)　这个算法输入关系序列 $j<k$,指出在某个偏序中对象 j 居前于对象 k,且假定 $1\leqslant j,k\leqslant n$。输出是嵌套在线性顺序下的 n 个对象的集合。所使用的内部表格是:QLINK[0],COUNT[1] = QLINK[1],COUNT[2] = QLINK[2],…,COUNT[n] = QLINK[n];TOP[1],TOP[2],…,TOP[n];一个存储池,它对于每个输入关系有一个节点,而且如上所示带有 SUC 和 NEXT 字段;用来引用存储池中的节点的链接变量 P;用来引用一个队列的前端和后端的整型变量 F 和 R,而这个队列的链接是在 QLINK 表格中;还有 N,是统计还有多少个对象有待输出的变量。

T1. [初始化] 输入 n 的值。对于 $1\leqslant k\leqslant n$,置 COUNT[k]←0 和 TOP[k]←Λ。置 N←n。

T2. [下个关系] 从输入中获取下个关系"$j<k$";然而如果输入已被穷尽,则转到 T4。

T3. [记录关系] COUNT[k]加 1。置 P⇐AVAIL，SUC(P)←k，NEXT(P)←TOP[j]，TOP[j]←P。(这是操作(8)。)转回 T2。

T4. [扫描零] (这时我们已经完成输入阶段;输入(18)现在将被转换成图 8 中显示的计算机表示。下一个工作是初始化输出的队列,它在 QLINK 字段中被链接在一起。)置 R←0 和 QLINK[0]←0。对于 $1 \leq k \leq n$,考察 COUNT[k],如果它为零,置 QLINK[R]←k 和 R←k。在对所有的 k 把这都做完了之后,置 F←QLINK[0](它将包含头一个遇到的使 COUNT[k]为零的 k 值)。

T5. [输出队列前端] 输出 F 的值。如果 F = 0,转到 T8;否则,置 N←N − 1,并且置 P←TOP[F]。(由于 QLINK 和 COUNT 表格重叠,我们有 QLINK[R] = 0;因此,当队列为空时条件 F = 0 出现。)

T6. [删除关系] 如果 P = Λ,转到 T7;否则 COUNT[SUC(P)]减 1,而且如果它由此而变成零,就置 QLINK[R]←SUC(P)和 R←SUC(P)。置 P←NEXT(P)并重复这一步骤。(我们正在从系统中对于某个 k 删去形如"F < k"的所有关系,并且当所有它们的前驱都已被输出时,把新节点放入队列中。)

T7. [从队列删除] 置 F←QLINK[F]和转回 T5。

T8. [过程结束] 本算法终止。如果 N = 0,我们已经以所求的"拓扑排序"输出所有对象的号码,且后面跟着一个零。否则 N 个还未输出的对象号码包含一个循环,同偏序的假设相违背。(关于打印出一个这样的循环的内容的算法,参见习题 23。) ▌

读者将发现对于输入(18)用手工尝试这个算法是有帮助的。算法 T 示出在顺序存储和链接存储技术之间很好的相互作用。对于主表格 X[1],…,X[n],使用的是顺序存储,这些表格包含 COUNT[k]和 TOP[k]的项,因为我们要在步骤 T3 中访问这个表格的"随机部分"。(然而,如果输入是字母型的,如同在第 6 章所述,为了更快的查找,将使用另一种类型的表格。)对于"直接后继"的表格使用链接存储,因为这些表格的表项在输入中没有特别的顺序。等候被输出的节点队列通过以输出的顺序把节点链接在一起而被保存在顺序表格的中间。这个链接是通过表格的下标而不是通过地址进行的;换句话说,当队列的前端是 X[k]时,我们有 F = k 而不是 F = LOC(X[k])。用在 T4,T6 和 T7 中的队列操作不同于在(14)和(17)中的那些操作,因为我们正利用在这个系统中的队列的特殊性质;在算法的这个部分无须建立节点或把节点返回到可用空间去。

以 MIX 汇编语言来对算法 T 进行编码有另外一些令人感兴趣之点。由于在这个算法中没有进行从表格中删除的操作(因为不必释放存储器以供以后使用),所以操作 P⇐AVAIL 可以如同在下面的行 19 和 32 中所示那样,以极其简单的方式来进行;我们无须保存任何存储器的链接池;而且我们可以连续地选择新的节点。这个程序按照上面提及的约定,通过使用磁带包括了完整的输入和输出,但为了简便起见忽略了缓冲区。读者不应对掌握在这个程序中的编码的细节感到有很大困难,因为它直接同算法 T 相对应。在这里还说明了变址寄存器的有效使用,它是链接存储器处理的一个重要方面。

程序 T(拓扑排序)　在这个程序中，应注意下列等价性：rI6≡N，rI5≡缓冲区指针，rI4≡k，rI3≡j 及 R，rI2≡AVAIL 及 P，TOP[j]≡X+j(4:5)，COUNT[k]≡QLINK[k]≡X+k(2:3)。

```
*   BUFFER AREA AND FIELD DEFINITIONS
COUNT      EQU    2:3                            字段符号名的定义
QLINK      EQU    2:3
TOP        EQU    4:5
SUC        EQU    2:3
NEXT       EQU    4:5
TAPEIN     EQU    1                              输入在带设备 1
TAPEOUT    EQU    2                              输出在带设备 2
BUFFER     ORIG   *+100                          带缓冲区域
           CON    -1                             缓冲区末尾的标志
* INPUT PHASE
TOPSORT    IN     BUFFER(TAPEIN)     1           T1. 初始化。输入头一个带块，
           JBUS   *(TAPEIN)                        等候完成
1H         LD6    BUFFER+1           1           N←n
           ENT4   0,6                1
           STZ    X,4                n+1         对于 0≤k≤n，置 COVNT[K]←0
           DEC4   1                  n+1           和 TOP[k]←Λ
           J4NN   *-2                n+1         (预处理 T4 步的 QLINK[0]←0)
           ENT2   X,6                1           在 X[n]之后可用存储开始
           ENT5   BUFFER+2           1           准备读头一对(j,k)
2H         LD3    0,5                m+b         T2. 下个关系
           J3P    3F                 m+b         j>0 吗？
           J3Z    4F                 b           输入穷尽了吗？
           IN     BUFFER(TAPEIN)     b-1         读出标志；读另一个带块区
           JBUS   *(TAPEIN)                        等候完成
           ENT5   BUFFER             b-1         恢复缓冲区指针
           JMP    2B                 b-1
3H         LD4    1,5                m           T3. 记录关系
           LDA    X,4(COUNT)         m           COUNT[k]
           INCA   1                  m             +1
           STA    X,4(COUNT)         m               →COUNT[k]
           INC2   1                  m           AVAIL←AVAIL+1
           LDA    X,3(TOP)           m           TOP[j]
           STA    0,2(NEXT)          m             →NEXT(P)
           ST4    0,2(SUC)           m           k←SUC(P)
```

```
              ST2    X,3(TOP)           m       P→TOP[j]
              INC5   2                  m       缓冲区指针增值
              JMP    2B                 m
4H            IOC    0(TAPEIN)          1       重绕输入带
              ENT4   0,6                1       T4. 扫描零。 k←n
              ENT5   -100               1       恢复输出的缓冲区指针
              ENT3   0                  1       R←0
4H            LDA    X,4(COUNT)         n       考察 COUNT[k]
              JAP    *+3                n       它非零吗?
              ST4    X,3(QLINK)         a       QLINK[R]←k
              ENT3   0,4                a       R←k
              DEC4   1                  n
              J4P    4B                 n       n≥k≥1
 * SORTING PHASE
              LD1    X(QLINK)           1       F←QLINK[0]
5H            JBUS   *(TAPEOUT)                 T5. 输出队列前端
              ST1    BUFFER+100,5       n+1     把 F 存入缓冲区域中
              J1Z    8F                 n+1     F 为零吗?
              INC5   1                  n       推进缓冲区指针
              J5N    *+3                n       检查缓冲器是否满了
              OUT    BUFFER(TAPEOUT)    c-1     如是,输出一个带块
              ENT5   -100               c-1     恢复缓冲区指针
              DEC6   1                  n       N←N-1
              LD2    X,1(TOP            n       P←TOP[F]
              J2Z    7F                 n       T6. 删除关系
6H            LD4    0,2(SUC)           m       rI4←SUC(P)
              LDA    X,4(COUNT)         m       COUNT[rI4]
              DECA   1                  m          -1
              STA    X,4(COUNT)         m            →COUNT[rI4]
              JAP    *+3                m       是否已经到达 0 了?
              ST4    X,3(QLINK)         n-a     如是,置 QLINK[R]←rI4
              ENT3   0,4                n-a     R←rI4
              LD2    0,2(NEXT)          m       P←NEXT(P)
              J2P    6B                 m       如果 P≠Λ,重复
7H            LD1    X,1(QLINK)         n       T7. 从队列中删除
              JMP    5B                 n       F←QLINK(F),转到 T5
8H            OUT    BUFFER(TAPEOUT)    1       T8. 过程结束
              IOC    0(TAPEOUT)         1       输出最后的块并重绕
```

```
          HLT    0,6          1      停机,在控制台上显示 N
X         END    TOPSORT             表格区域开始  ▌
```

借助于基尔霍夫定律对算法 T 进行分析是十分简单的;执行时间有近似形式 $c_1m + c_2n$,其中 m 是输入关系个数,n 是对象的个数,c_1 和 c_2 是常数。很难想像对于这个问题还有更快的算法!在这个分析中精确的量在上面的程序 T 中给出,其中 a = 没有前驱的对象的个数,b = 输入带记录的个数 $= \lceil (m+2)/50 \rceil$,以及 c = 输出中带记录的个数 $= \lceil (n+1)/100 \rceil$。排除输入输出操作,在这种情况下总的运行时间仅是 $(32m + 24n + 7b + 2c + 16)u$。

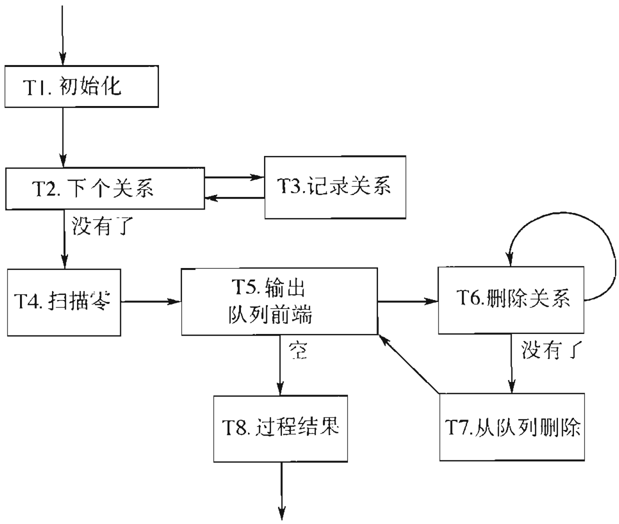


图 9　拓扑排序

类似于算法 T(但没有队列链接的重要特性)的一个拓扑排序技术首先是由 A. B. Kahn 发表的,*CACM* **5** (1962),558 ~ 562。一个偏序的拓扑排序总是可能的这一事实首先以文字方式加以证明是由 E. Szpilrajn 完成的,*Fundamenta Mathematica* **16** (1930),386 ~ 389;他对于无穷集合和有限集合都进行了证明,并且指出这一结果已经为他的很多同事所知晓了。

尽管算法 T 如此有效率,但在 7.4.1 小节,我们还将研究一个更好的拓扑排序算法。

习　题

▶ 1. [*10*] 弹出栈的操作(9)提及 UNDERFLOW(下溢)的可能性,为什么压入栈的操作(8)未提及 OVERFLOW(上溢)的可能性?

2. [*22*] 写出进行插入操作(10)的一个"通用"的 MIX 子程序。这个子程序应当有下列特征描述(如同在 1.4.1 小节一样):

调用序列:JMP　INSERT　转到子程序
　　　　NOP　T　　　指针变量的地址

入口条件:rA = 要插入一个新节点的 INFO 字段的信息。

出口条件:指针是链接变量 T 的栈在栈顶上有新节点;rI1 = T;rI2,rI3 被改变。

3. [*22*] 写出进行删除操作(11)的一个"通用"的 MIX 子程序。这个子程序应当有下列的特征描述:

调用序列:JMP　DELETE　转到子程序
　　　　NOP　T　　　指针变量的地址
　　　　JMP UNDERFLOW　如果读出 UNDERFLOW,第一出口

入口条件:无。

出口条件:如果指针是链接变量 T 的栈为空,则采用第一出口;否则删除该栈的顶部节点,并且在"JMP DELETE"指令下面第三个单元出口。在后一种情况下 rI1 = T 和 rA 是被删除的节点 INFO 字段的内容。在两种情况下,rI2 和 rI3 都为这个子程序所使用。

4. [*22*] (10)中的程序是基于操作 P⇐AVAIL 写成的,如同在(6)中给出的那样。说明如何写一个 OVERFLOW 的子程序,使得无须对(10)的编码作任何改动,如同由(7)所给出的那样,操作 P⇐AVAIL 利用 SEQMIN。为保持通用性,除了 rJ 和可能还有比较指示器之外,子程序不应改变任何寄存器的内容。它应出口到单元 rJ - 2,而不是通常的 rJ。

▶ 5. [*24*] 操作(14)和(17)给出一个队列的效果;说明如何进一步定义操作"插入到前端",以便得到一个输出受限的双端队列的所有动作。如何定义"从后端删除"的操作(使得我们将有一个一般的双端队列)?

6. [*21*] 在操作(14)中我们置 LINK(P)←Λ,而在队列后端的紧接着的插入将改变这同一个链接字段的值。说明如果对(17)中的测试"F = Λ"作一改动,如何可以避免(14)中对 LINK(P)的设置?

▶ 7. [*23*] 试设计一个算法,"逆转"一个像(1)中那样的链接线性表,即改变它的链接使得诸项以相反的顺序出现。[例如,如果表(1)被"逆转",我们将有 FIRST 链接到包含项 5 的节点;该节点将链接到包含项 4 的一个节点,等等。]假定诸节点有(3)的形式。

8. [*24*] 试写出对于习题 7 的问题的一个 MIX 程序,并尝试把你的程序设计得尽可能地快。

9. [*20*] 下列关系中哪一个是在特定集合 S 上的一个偏序?[注:如果如下定义的是关系"$x \prec y$",则本题旨在定义关系"$x \preceq y \equiv (x \prec y$ 或 $x = y)$",而后确定 $\preceq$ 是否为一个偏序。](a) S = 所有有理数,$x \prec y$ 指 $x > y$。(b) S = 所有人,$x \prec y$ 指 x 是 y 的一个祖先。(c) S = 所有整数,$x \preceq y$ 指 x 是 y 的一个倍数(即 $x \bmod y = 0$)。(d) S = 在本文中证明的所有数学结果,$x \preceq y$ 指 y 的证明依赖于 x 的正确性。(e) S = 所有正整数,$x \preceq y$ 指 $x + y$ 为偶数。(f) S = 一个子程序集合,$x \prec y$ 指"x 调用 y",即 x 处于操作中时,y 可能也在操作,但不允许递归。

10. [*M21*] 给定"$\subset$"是满足偏序的性质 i)和 ii)的一个关系,证明,由规则"$x \preceq y$ 当且仅当 $x = y$ 或 $x \subset y$"定义的关系"$\preceq$"满足偏序的所有三个性质。

▶ 11. [*24*] 拓扑排序的结果并不总是完全确定的,因为可能有好几种方法来安排节点并且满足拓扑排序的条件。找出所有可能的方式来把图 6 的节点安排成拓扑顺序。

12. [*M20*] 一个 n 个元素的集合有 2^n 个子集,而且按照集合包含关系这些子集是偏序的。给出把这些子集安排成拓扑顺序的两个有趣方式。

13. [*M48*] 有多少种方法来把习题 12 中的 2^n 个子集安排成拓扑顺序?(把答案作为 n 的

函数给出。)

14. [*M21*] 一个集合 S 的一个线性序，也叫做一个全序，是满足下列附加的“可比较性”条件的偏序。

iv)对 S 中的任何两个对象 x 和 y，或者 $x \preceq y$ 或者 $y \preceq x$。

直接地从给出的定义证明，当且仅当关系 $\preceq$ 是一个线性序时，一个拓扑排序能得到惟一可能的输出。(你可以假定集合 S 是有限的。)

15. [*M25*] 证明对于一个有限集合 S 上的任何偏序，像在(18)和图 6 中一样，存在表征这个序的非冗余关系的惟一的集合。当 S 是一个无穷集合时，同样的事实是否也成立？

16. [*M22*] 给定一个集合 $S = \{x_1, \cdots, x_n\}$ 上的任何偏序，我们可以构造它的伴随矩阵(a_{ij})，如果 $x_i \preceq x_j$ 则 $a_{ij} = 1$，否则 $a_{ij} = 0$。证明有一个方法来排列这个矩阵的行和列，使得对角线之下的所有项全为零。

▶ 17. [*21*] 如果给算法 T 提供输入(18)，它将产生什么输出？

18. [*20*] 当算法 T 结束时，QLINK[0]，QLINK[1]，…，QLINK[n]的值有什么意义，如果有的话？

19. [*18*] 在算法 T 中，我们在步骤 T5 中考察队列的前端位置，但在步骤 T7 之前不从队中删去该元素。如果我们在步骤 T5 的结尾处而不是在 T7 中置 F←QLINK[F]，那将发生什么情况？

▶ 20. [*24*] 算法 T 使用 F，R 和 QLINK 表格来得到一个队列的效果，该队列包含其 COUNT 字段已经变成零但其后继关系还未被删除的那些节点。一个栈可否取代队列用于此目的？如果可以，试把得到的算法同算法 T 进行比较。

21. [*21*] 如果关系“$j < k$”之一在输入中被重复多次，算法 T 仍将实现正确的拓扑排序吗？

22. [*23*] 程序 T 假定它的输入带包含正确的信息，但一个打算通用的程序总是要对它的输入作仔细的检查使得可以发现笔误，并且使得程序不能“自损”。例如，如果对于 k 的输入关系之一是负的，则当存入 X[k]时程序 T 可能错误地改变它自己的指令之一。试提出修改程序 T 的建议使得它适合于普遍的使用。

▶ 23. [*27*] 当由于拓扑排序算法已经探测出输入中的一个循环(见步骤 T8)而不能进行下去时，停止下来并说“有一个循环”通常是毫无用处的。有帮助的是把循环之一打印出来，由此显示出部分输入是有错的。试推广算法 T 使得它在必要时将进行对于一个循环的额外打印。[提示：正文在步骤 T8 中给出当 N > 0 时一个循环存在的证明；该证明提示了一个算法。]

24. [*24*] 把习题 23 中所做的对算法 T 的扩充加入到程序 T 中。

25. [*47*] 试设计出一个尽可能有效的算法，它对非常大的集合 S 进行拓扑排序，这个集合含有比计算机内存所能容纳的数量还要大得多的节点。假定输入、输出以及临时工作空间都通过磁带进行。[可能的提示：对输入的传统排序允许我们假定，对于一个给定节点，其所有关系一起出现。但那样一来可以做什么呢？特别是，我们必须考虑最坏的情况，其中给定的序已经是被杂乱地排列的一个线性序；在第 5 章引言中的习题 24 说明了通过对数据进行 $O(\log n)^2$ 次扫描如何来处理这个情况。]

26. [*29*] (子程序的分配) 假设我们有适用于一台 20 世纪 60 年代风格的计算机装置的以浮动形式包含主子程序库的一个带。装入程序要确定所用的每个子程序浮动的数量，使得它可以对整个带进行一次扫描来装入必要的程序。问题在于，有些子程序要求其它子程序也存在于内存中。不经常使用的子程序(它们存放在靠近带的末尾)可能调用经常使用的子程序(它们存放在靠近带的开始处)。因此在通过整个带之前，我们要知道需要的所有子程序。

对付这个问题的一个方法是有一个装入到内存中的“带目录”。装入程序可以访问两个表格：

a)带目录。这表格由有下列形式的可变长的节点组成：

B	SPACE	LINK
B	SUB1	SUB2
	⋮	
B	SUBn	0

或

B	SPACE	LINK
B	SUB1	SUB2
	⋮	
B	SUB($n-1$)	SUBn

其中 SPACE 是此子程序所需要的内存的字数；LINK 是对于出现在这个子程序下面的带上的子程序的目录项的一个链接；SUB1，SUB2，…，SUBn ($n\geqslant 0$)是这个子程序所需要的任何其它子程序的目录项的链接；除了最后一个字外，在所有的字上 B = 0，在一个节点的最后一个字上，B = −1。库带上的头一个子程序的目录项的地址由链接变量 FIRST 确定。

b)要装入的直接由程序访问的子程序清单。它保存在连续的单元 X[1]，X[2]，…，X[N]中，其中 N≥0 是为装入程序所已知的一个变量。在这个表中的每个表项是所要求的子程序的目录表项的一个链接。

装入程序也知道 MLOC，即装入的头一个子程序要使用的浮动的数量。

作为一个小的例子，考虑下列配置：

带目录	B	SPACE	LINK	所需子程序清单
1000	0	20	1005	X[1] = 1003
1001	−1	1002	0	X[2] = 1010
1002	−1	30	1010	
1003	0	200	1007	N = 2
1004	−1	1000	1006	FIRST = 1002
1005	−1	100	1003	MLOC = 2400
1006	−1	60	1000	
1007	0	200	0	
1008	0	1005	1002	
1009	−1	1006	0	
1010	−1	20	1006	

在这情况下带目录说明，在带上的子程序依次是 1002，1010，1006，1000，1005，1003 和 1007。子程序 1007 占用 200 个单元并且意味着使用子程序 1005，1002 和 1006 等。要装入的程序需要子程序 1003 和 1010，它们要被放置到大于等于 2400 的单元中。这些子程序转过来意味着也必须装入 1000，1006 和 1002。

子程序的分配程序将改变 X 表格使得每个项 X[1]，X[2]，…有下列形式(最后一项除外，它在下面说明)：

+	0	BASE	SUB

其中 SUB 是要装入的子程序，而 BASE 是浮动的数量。这些项要按子程序在带上出现的顺序出现。对于上述例子，一个可能的答案将是

	BASE	SUB		BASE	SUB
X[1]:	2400	1002	X[4]:	2510	1000
X[2]:	2430	1010	X[5]:	2530	1003
X[3]:	2450	1006	X[6]:	2730	0

最后一项包含头一个未使用的内存地址。

(显然,这不是处理子程序库的惟一方法。设计一个库的适当方法高度地依赖于所用的计算机以及要加以处理的应用。大型现代计算机要求全然不同的处理子程序库的方法。但是无论如何这是一个好的习题,因为它涉及对于顺序数据和链接数据的有趣操作。)

本习题中的问题是对于所述的任务设计一个算法。你的分配程序可以在准备它的答案时以任何方式转换带目录,因为带目录可以在它的下个赋值时由子程序分配程序重新读人,而且带目录不为装入程序的其它部分所需。

27. [*25*] 对习题 26 的子程序分配算法写出一个 MIX 程序。

28. [*40*] 下列构造显示如何"解决"相当普通的两人游戏类型,包括国际象棋,拿子游戏(nim)以及许多较简单的游戏:考虑有限的节点集合,它们每一个表示游戏中可能的位置。对于每个位置,存在把该位置转换成某个其它位置的零个或更多的移动。我们说位置 x 是位置 y 的一个前驱(以及 y 是 x 的一个后继),如果有从 x 到 y 的一个移动的话。没有后继的某些位置被分类作为*胜利*位置或*失败*位置。移入位置 x 处的游戏者是移人位置 x 的后继处的游戏者的对手。

给定位置的这样一个格局,我们可以计算胜利位置的整个集合(它就是那样一些位置,即另一个游戏者的移动可赢得胜利),以及计算失败位置的集合(它就是那样一些位置,即游戏者面对一个高手必然失败)。办法是重复地执行下列操作直到它不产生改变为止:如果一个位置的所有后继都被标记为"胜",则标记这个位置为"失败";如果它的后继中至少有一个被标记为"失败",则把这个位置标记为"胜利"。

在这个操作被重复了尽可能多次之后,可能会有某些位置,它们仍然未被标识;在这样一个位置中一个游戏者既不能赢得胜利也不能被迫失败。

得到胜利和失败位置的完全集合的这一过程可以被修改成非常类似算法 T 的--个有效的计算机算法。对于每个位置,我们可以保持还未被标记为"胜利"的它的后继个数的计数,以及所有它的前驱的一张表。

本习题中的问题是写出还仅仅模糊地描述的这个算法的细节,并且把它应用于不涉及太多可能位置的某些有趣的游戏[比如"军棋":É. Lucas, *Récréations Mathématiques* **3**(Paris:1893),105~116;E.R.Berlekamp,J. H. Conway 及 R. K. Guy,*Winning Ways* **2** (Academic Press,1982),第 21 章]。

▶ 29. [*21*] (a)通过把所有节点都放在 AVAIL 栈上,仅仅给定 FIRST 的值,试给出一个算法来整个"擦掉"像(1)那样的表。这个算法应当尽可能快地操作。(b)给出 F 和 R 的值,对于像(12)那样的表重复(a)。

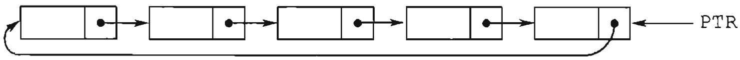
30. [*17*] 假设队列像在(12)中那样表示,但以 F = Λ 和 R *无定义*来表示一个空队列。什么插入和删除过程将代替(14)和(17)?

2.2.4 循环表

在链接方式上的稍微变化就为我们提供了对于上一小节的方法的重要改变。

一个循环链接表(简称为循环表)有这样一个性质,即它的最后的节点链接到头一个节点而不是 Λ。于是,从任何给定的点开始,有可能访问整个表;我们也实现了额外程度的对称性,而且如果愿意,我们不必想像这个表有一个最后或最开头的节点。

下列状况是典型的:



(1)

假定诸节点有两个字段,INFO 和 LINK,如同在上一小节所述那样。有一个链接变量 PTR,它指向表最右边的节点,而 LINK(PTR)是最左边的节点的地址。下列的原始操作是最重要的:

a)把 Y 插入左边: P⇐AVAIL, INFO(P)←Y,LINK(P)←LINK(PTR), LINK(PTR)←P。

b)把 Y 插入右边: 把 Y 插入左边,然后 PTR←P。

c)把 Y 置成左节点并删除: P←LINK(PTR),Y←INFO(P),LINK(PTR)←LINK(P),AVAIL⇐P。

乍一看,操作 b)有点令人惊讶;但在框图(1)中操作 PTR←LINK(PTR)有效地把最左的节点移动到右边去,而如果把表当做一个循环而不是当做有相连端点的一条直线,这是十分容易理解的。

警觉的读者将发现,我们在操作 a),b)和 c)中已经犯了一个严重的错误。什么错误?答案:我们已经忘了考虑一个空表的可能性。例如,如果把操作 c)应用于表(1)五次,我们将有 PTR 指向 AVAIL 表中的一个节点,而这会导致严重的困难;例如,想像再次应用操作 c)! 如果我们取这样一个位置,即在空表的情况下 PTR 将等于 Λ,我们可以通过插入附加的指令"如果 PTR = Λ,则 PTR←LINK(P)←P; 否则…"于 a)中的"INFO(P)←Y"之后,来修改这些操作; 在 c)之前测试"如果 PTR = Λ,则 UNDERFLOW"; 而在 c)之后测试"如果 PTR = P,则 PTR←Λ"。

注意操作 a),b)和 c)在 2.2.1 小节的意义下给了我们一个输出受限双端队列的动作。因此,我们特别地发现,一个循环表可以用做一个栈或者一个队列。组合的操作 a)和 c)给了我们一个栈;操作 b)和 c)给了我们一个队列。这些操作仅仅比上一小节中的对应操作稍微不那么直接而已;在那里我们看到可以使用两个指针 F 和 R 来对线性表实施操作 a),b)和 c)。

还有其它重要操作,通过使用循环表而变得有效率。例如,"擦除"一个表很方便,即,一次就把整个循环表放到 AVAIL 栈上:

$$\text{如果 PTR} \neq \Lambda \text{ 则 AVAIL} \leftrightarrow \text{LINK(PTR)} \qquad (2)$$

[回忆一下,↔操作表示互换:P←AVAIL,AVAIL←LINK(PTR),LINK(PTR)←P。]如果 PTR 指向循环表的任何地方,操作(2)显然正确。然后我们当然应置 PTR←Λ。

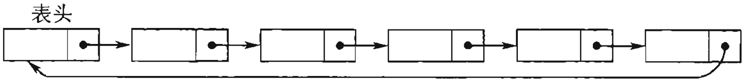
使用一个类似的技术,如果 PTR_1 和 PTR_2 分别指向不相交的循环表 L_1 和 L_2,我们可以把整个表 L_2 插到 L_1 的右边:

如果 $PTR_2 \neq \Lambda$,则
　　(如果 $PTR_1 \neq \Lambda$,则 LINK(PTR_1)↔LINK(PTR_2);　　(3)
　　置 $PTR_1 \leftarrow PTR_2$, $PTR_2 \leftarrow \Lambda$)

以各种方法把一个循环表分成两个是可以实施的另一种简单的操作。这些操作对应于串的连接和拆分。

因此我们看到一个循环表不仅可用于表示固有的循环结构,而且还可以表示线性结构;一个具有指向末端节点指针的循环表实质上等价于一个具有指向前端和末端两个指针的线性表。同这个发现相关联很自然要问的问题是:"当存在有循环对称性时,我们如何找到这个表的结尾?"因为没有 Λ 链接来标志结尾了! 答案是,我们对整个表进行操作,即从一个节点移动到另一个节点时,如果我们回到了起始位置,就应当停止(当然假定起始位置还在这个表中)。

对刚才提出的问题的另一个解,是在每个循环表中放置一个特殊的可识别的节点,作为一个方便的停止位置。这个特殊的节点,称做*表头*,而且在应用中我们通常都会发现,坚持每一个循环表恰有一个节点作为其表头十分方便。一个优点是,循环表因而将绝不成为空的。有了表头,图(1)变成



(4)

对于像表(4)这样的表的访问通常都是通过表头进行的。表头通常都在一个固定的内存单元中。表头的缺点是没有指向右端的指针,因此我们必须牺牲上面所述的操作b)。

可以把(4)同上一小节开始处的 2.2.3-(1)作比较,其中同"项 5"相关联的链接现在指向 LOC(FIRST)而不是 Λ;变量 FIRST 现在被认为是在一个节点内的链接,即在 NODE(LOC(FIRST))中的链接。(4)和 2.2.3-(1)之间的主要区别是,(4)使得有可能(尽管未必很有效率)从任何其它点到达表的任何点。

作为使用循环表的一个例子,我们将讨论对变量 x, y 和 z 且带有整系数的多项式算术。有许多问题科学家都要对多项式而不仅仅是数进行操作。我们考虑的是像

$$(x^4+2x^3y+3x^2y^2+4xy^3+5y^4)\text{ 乘以 }(x^2-2xy+y^2)$$

来得到

$$(x^6-6xy^5+5y^6)$$

这样的乘法运算。链接分配是用于这一目的的自然工具,因为多项式可以增长到不可预料的大小,而且同时我们要在内存中表示许多多项式。

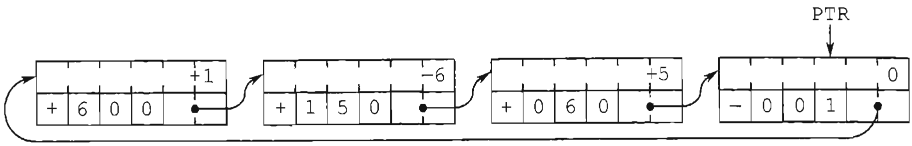
这里我们将考虑加法和乘法两个运算。我们假设,把一个多项式表示为每个节点代表一个非零项的表,而且每个节点有双字形式

COEF				
±	A	B	C	LINK

(5)

这里 COEF 是 $x^A y^B z^C$项的系数。我们将假定系数和指数总是在这个格式所允许的范围

中,因而在计算期间无须对这个范围进行检验。ABC 的记号将被用来代表节点(5)的 ±A B C 字段,把它当做一个单位。ABC 的符号,即(5)中第二个字的符号,将总为正,除了在每个多项式的末尾有一个特殊节点,即 ABC = -1 和 COEF = 0 之外。类似于上面我们关于表头的讨论,这个特殊的节点是一个很大的便利,因为它提供了一个方便的标志,因而避免了空表的问题(对应于多项式 0)。如果我们沿着链接的方向,除了特殊节点(它有 ABC = -1)链接到 ABC 的最大值之外,表的节点是以 ABC 字段的递减顺序出现。例如,多项式($x^6 - 6xy^5 + 5y^6$)将被表示如下:



算法 A(*多项式加法*) 假定 P 和 Q 是指向上述形式的多项式的指针变量。这个算法把多项式(P)加到多项式(Q)上。表 P 将不变;表 Q 将返回和。在本算法结束时,指针变量 P 和 Q 返回到它们的起始点处,还使用辅助指针变量 Q1 和 Q2。

A1. [初始化] 置 P←LINK(P),Q1←Q,Q←LINK(Q)。(现在 P 和 Q 两者都指向它们的多项式的前导项。在 Q = LINK(Q1)的意义下,贯穿于本算法的大部分时间,变量 Q1 将落后于 Q 一步。)

A2. [ABC(P):ABC(Q)] 如果 ABC(P) < ABC(Q),则置 Q1←Q,Q←LINK(Q)并且重复这一步。如果 ABC(P) = ABC(Q),转到步骤 A3。如果 ABC(P) > ABC(Q),转到步骤 A5。

A3. [系数相加] (我们已经找到有相同指数的项。)如果 ABC(P) < 0,则算法终止。否则置 COEF(Q)←COEF(Q) + COEF(P)。现在如果 COEF(Q) = 0,转到 A4;否则,置 P←LINK(P),Q1←Q,Q←LINK(Q),并转到 A2。(奇怪,后面这些操作和步骤 A1 相同。)

A4. [删除为零项] 置 Q2←Q,LINK(Q1)←Q←LINK(Q),而且 AVAIL⇐Q2。(在步骤 A3 中建立起来的零项已从多项式(Q)中删去。)置 P←LINK(P)并转回到 A2。

A5. [插入新项] (多项式(P)包含有在多项式(Q)中不存在的项,所以我们把它插入到多项式(Q)中。) 置 Q2⇐AVAIL,COEF(Q2)←COEF(P),ABC(Q2)←ABC(P),LINK(Q2)←Q,LINK(Q1)←Q2,Q1←Q2,P←LINK(P),并返回步骤 A2。 ▌

算法 A 最值得注意的特征之一是指针变量 Q1 沿着表跟踪指针 Q 的方式。这是表处理算法非常典型的方式,而且我们将看到具有同一个特征的十余个算法。读者是否看出为什么在算法 A 中使用这个思想?

对链接表经验不多的读者将会发现,仔细地研究算法 A 是非常有教益的;作为检验的例子,试做 $x + y + z$ 和 $x^2 - 2y - z$ 的加法。

给出了算法 A,乘法运算是令人惊异地容易的:

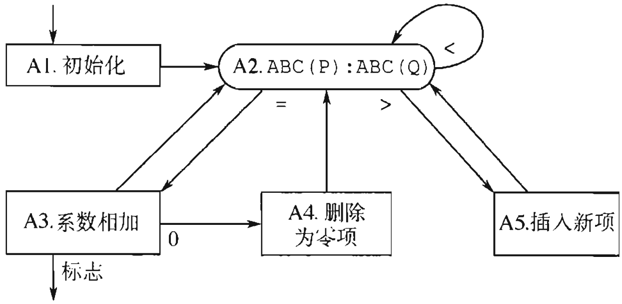


图 10　多项式加法

算法 M(多项式乘法)　类似于算法 A,这个算法以

多项式(Q) + 多项式(M) × 多项式(P)

来代替多项式(Q)。

M1. [下一个乘数]　置 M←LINK(M)。如果 ABC(M) < 0,则算法终止。

M2. [乘法循环]　除了每当记号"ABC(P)"出现于该算法时,就以"(如果 ABC(P) < 0,则 − 1,否则 ABC(P) + ABC(M))"代替它之外,都实施算法 A;每当"COEF(P)"出现于该算法中时,就以"COEF(P) × COEF(M)"代替它,然后转回到步骤 M1。 ▌

以 MIX 语言做算法 A 的程序设计再次显示在一台计算机上对链接表的操作是很容易的。在以下的程序代码中,我们假定 OVERFLOW 是(由于缺乏可用存储空间而)终止程序,或者进一步寻找可用空间并出口到 rJ − 2 处的一个子程序。

程序 A(多项式加法)　这个子程序被写成使得它可以和乘法子程序一起使用(参见习题 15)。

调用序列:JMP ADD。

入口条件:rI1 = P,rI2 = Q。

出口条件:多项式(Q)已被多项式(Q) + 多项式(P)所代替;rI1 和 rI2 不变;所有其它寄存器的内容未定义。

在下列的程序代码中,在算法 A 的记号下,P ≡ rI1,Q ≡ rI2,Q1 ≡ rI3,Q2 ≡ rI6。

```
LINK  EQU   4:5                      LINK 字段的定义
ABC   EQU   0:3                      ABC 字段的定义
ADD   STJ   3F            1          子程序的入口
1H    ENT3  0,2           1 + m''    A1. 初始化。置 Q1←Q
      LD2   1,3(LINK)     1 + m''    Q←LINK(Q1)
0H    LD1   1,1(LINK)     1 + p      P←LINK(P)
SW1   LDA   1,1           1 + p      rA(0:3)←ABC(P)
2H    CMPA  1,2(ABC)      x          A2. ABC(P):ABC(Q)
      JE    3F            x          如果相等转到 A3
```

```
          JG    5F           p' + q'   如果大于,转到 A5
          ENT3  0,2          q'        如果小于,则置 Q1←Q
          LD2   1,3(LINK)    q'        Q←LINK(Q1)
          JMP   2B           q'        重复
3H        JAN   *            m + 1     A3. 系数相加
SW2       LDA   0,1          m         COEF(P)
          ADD   0,2          m            + COEF(Q)
          STA   0,2          m              →COEF(Q)
          JANZ  1B           m         如果非零则转移
          ENT6  0,2          m'        A4. 删除为零的项。Q2←Q
          LD2   1,2(LINK)    m'        Q←LINK(Q)
          LDX   AVAIL        m'        ⎫
          STX   1,6(LINK)    m'        ⎬AVAIL⇐Q2
          ST6   AVAIL        m'        ⎭
          ST2   1,3(LINK)    m'        LINK(Q1)←Q
          JMP   0B           m'        转去推进 P
5H        LD6   AVAIL        p'        ⎫
          J6Z   OVERFLOW     p'        ⎪A5. 插入新的项
          LDX   1,6(LINK)    p'        ⎬Q2⇐AVAIL
          STX   AVAIL        p'        ⎭
          STA   1,6          p'        ABC(Q2)←ABC(P)
SW3       LDA   0,1          p'        rA←COEF(P)
          STA   0,6          p'        COEF(Q2)←rA
          ST2   1,6(LINK)    p'        LINK(Q2)←Q
          ST6   1,3(LINK)    p'        LINK(Q1)←Q2
          ENT3  0,6          p'        Q1←Q2
          JMP   0B           p'        转去推进 P     ▌
```

注意算法 A 仅对两个表的每一个遍历一次而已,不需要循环好多遍。使用基尔霍夫定律,我们发现,对于指令计数的分析不存在困难;执行时间依赖于四个量

m' = 相互抵消的匹配项的数目

m'' = 不能消去的匹配项的数目

p' = 在多项式(P)中未匹配的项的数目

q' = 在多项式(Q)中未匹配的项的数目

对程序 A 给出的分析使用下列缩写:

$$m = m' + m'',\ p = m + p',\ q = m + q',\ x = 1 + m + p' + q'$$

对于 MIX 来说运行时间是$(27m' + 18m'' + 27p' + 8q' + 13)u$。在本算法执行期间所需

要的存储池中的节点的总数至少是 $2+p+q$，至多是 $2+p+q+p'$。

习　题

1. [*21*] 在本小节开始处正文提出，一个空的循环表可以由 PTR = Λ 来表示。而由 PTR = LOC(PTR)表示一个空表可能和循环表的原理更为一致。这个约定便于在本小节开始处描述的操作 a)，b)或 c)的实施吗？

2. [*20*] 假定 PTR_1 和 $\text{PTR}_2 \neq \Lambda$，试画一个"之前和之后"的框图说明连接操作(3)的效果。

▶ 3. [*20*] 如果 PTR_1 和 PTR_2 两者都指向同一个循环表的节点，操作(3)做什么？

4. [*20*] 使用框图(4)，指出给出一个栈的效果的插入和删除操作。

▶ 5. [*21*] 试设计取如同(1)那样的循环表而把所有箭头的方向颠倒过来的一个算法。

6. [*18*] 试给出对于多项式(a) $xz-3$；(b) 0 的表的表示框图。

7. [*10*] 为什么假定一个多项式表的 ABC 字段以递减顺序出现是有用的？

▶ 8. [*10*] 为什么在算法 A 中令 Q1 比 Q 拖后一步是有用的。

▶ 9. [*23*] 如果 P = Q(即两个指针变量指向同一个多项式)算法 A 能正确工作吗？如果 P = M 或 P = Q 或 M = Q，算法 M 能正确工作吗？

▶ 10. [*20*] 在本小节的诸算法假定，在多项式中我们使用三个变量 x，y 和 z，并且假定它们的指数都不会超过 $b-1$(其中 b 是在 MIX 的情况下字节的大小)。假定代替的是，我们要做只有一个变量 x 的多项式的加法和乘法，而且令它的指数可取达 b^3-1 的值，则对算法 A 和 M 应作什么改动？

11. [*24*] (本题和以下许多题的目的是建立和程序 A 一起的，对多项式算术运算有用的子程序包。)由于算法 A 和 M 改变多项式(Q)的值，有时有一个制作一个给定多项式的副本的子程序是有用的。试写出有下列特征描述的一个 MIX 子程序。

调用序列：JMP COPY。

入口条件：rI1 = P。

出口条件：rI2 指向新近建立的等于多项式(P)的一个多项式；

rI1 不变；其它寄存器无定义。

12. [*21*] 试比较习题 11 中的程序和当多项式(Q) = 0 时程序 A 的运行时间。

13. [*20*] 按照下列特征描述写一个 MIX 程序：

调用序列：JMP ERASE。

入口条件：rI1 = P。

出口条件：多项式(P)已被加到 AVAIL 表上；所有寄存器的内容无定义。

[注：这个子程序可以同习题 11 的子程序以序列"LD1 Q；JMP ERASE；LD1 P；JMP COPY；ST2 Q"一起使用来实现"多项式(Q)←多项式(P)"的效果。]

14. [*22*] 按照下列特征描述写一个 MIX 子程序：

调用序列：JMP ZERO。

入口条件：无。

出口条件：rI2 指向新建立的等于 0 的多项式；其它寄存器无定义。

15. [*24*] 按照下列特征描述，写出一个实现算法 M 的 MIX 子程序：

调用序列：JMP MULT。

入口条件：rI1 = P，rI2 = Q，rI4 = M。

出口条件：多项式(Q)←多项式(Q) + 多项式(M) × 多项式(P)；rI1，rI2，rI4 不变；其它寄存器无定义。

(注：使用程序 A 作为子程序，并且改变 SW1，SW2 和 SW3 的设置。)

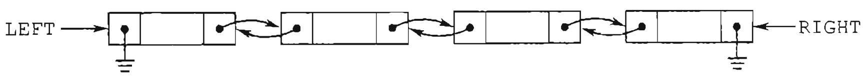
16. [*M28*] 借助于某些相关的参数，试估计习题 15 中的子程序的运行时间。

▶ 17. [*22*] 不使用上一小节那样以 Λ 终止的直线式线性链接表，而是如本小节这样，使用循环表来表示多项式，有什么优点？

▶ 18. [*25*] 试设计在一台计算机内表示循环表的方法，在这个方法下，可以有效地以两个方向遍历表，而且每个节点仅使用一个链接字段。[提示：如果有对两个连续节点 x_{i-1} 和 x_i 的两个指针，应当有可能对 x_{i+1} 和 x_{i-2} 这两者定址。]

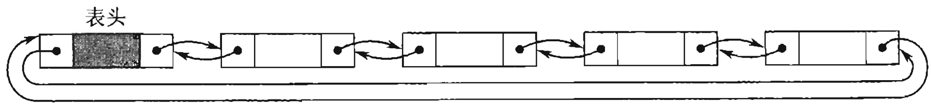
2.2.5 双重链接表

为了在线性表的操作中获得更大的灵活性，我们可以在每个节点中包括两个链接，指向该节点两边的项：



(1)

这里 LEFT 和 RIGHT 是指向表的左边和右边的指针变量。表的每一个节点有两个链接，可称为 LLINK 和 RLINK。通过这样一个表示容易实现一般双端队列的操作；见习题 1。然而，如果像上一小节所描述的那样，一个表头节点是每个表的一部分，则双重链接表的操作几乎总是变得更为容易。当存在一个表头时，我们有以下双重链接表的典型框图：



(2)

表头的 RLINK 和 LLINK 字段取(1)中的 LEFT 和 RIGHT 的位置。左和右之间完全对称；表头同样也可以表示在(2)的右边。如果表为空，则表头的两个字段指向这个表头本身。

如果 X 是在表(2)(包括表头在内)中的任何节点的位置，则它显然满足条件

$$\text{RLINK(LLINK(X))} = \text{LLINK(RLINK(X))} \qquad (3)$$

这个事实是(2)比(1)更可取的主要原因。

一个双重链接表通常比单链接表占用更多的存储空间(尽管在未完全填满一个计算机字的节点中有时还有供另一个链接的空间)。但是对双向链接能够有效地实施的额外操作通常是对额外空间要求的丰厚补偿。除了当考察一个双重链接表时能够随意地向前和向后移动这一明显的优点外，主要的新能力之一是这样一个事实，即*仅仅给出 X 的值，我们就能从它所在表中删除* NODE(X)。这一删除操作很容易从“之前和之后”框图(图 11)导出，而且很简单：

$$\begin{aligned}&\text{RLINK(LLINK(X))} \leftarrow \text{RLINK(X)},\\ &\text{LLINK(RLINK(X))} \leftarrow \text{LLINK(X)},\ \text{AVAIL} \Leftarrow \text{X}\end{aligned} \tag{4}$$

在一个仅有单向链接的表中，如果不知道在链接中哪一个节点居于 NODE(X)之前，我们就不能删除它，因为当 NODE(X)被删除时，居前的节点需要改变其链接。在 2.2.3 和 2.2.4 小节所考虑的所有算法中，只要一个节点要被删除，就可了解到这一点；特别是，参看算法 2.2.4A，其中我们有恰好为此目的的跟随指针 Q 的指针 Q1。但是我们将遇到若干个算法，它们要求从一个表的中间随机地删去节点。双重链接表就是由于这样一个原因而经常被使用的。（我们将指出，在一个循环表中，给定 X，如果遍历整个循环表来寻找 X 的前驱，则有可能删除 NODE(X)。但当这个表很长时，这种操作显然是低效的，所以要它替代双重链接表是难以接受的。也请参看习题 2.2.4-8 的答案。）

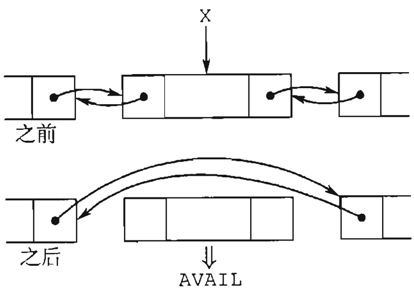


图 11　从一个双重链接表进行删除

类似地，一个双重链接表允许在左边或右边邻近于 NODE(X)处很容易地插入一个新节点。步骤

$$\begin{aligned}&\text{P} \Leftarrow \text{AVAIL},\ \text{LLINK(P)} \leftarrow \text{X},\ \text{RLINK(P)} \leftarrow \text{RLINK(X)},\\ &\text{LLINK(RLINK(X))} \leftarrow \text{P},\ \text{RLINK(X)} \leftarrow \text{P}\end{aligned} \tag{5}$$

就是在 NODE(X)右边来做这样一个插入的；通过交换左和右我们可以得到在左边插入的对应算法。操作(5)改变五个链接的设置，因此它要比在单向表中的插入操作慢一点，因为在那里只需要改变三个链接。

作为使用双重链接表的一个例子，我们来考虑编写一个**离散模拟程序**。“离散模拟”指的是对于这样一个系统的模拟，即在这个系统状态中的所有变化都可以假定是在某些离散时刻发生的。被模拟的“系统”通常是一组大部分独立的个别的活动，尽管这些活动彼此交互作用；这方面的例子包括在一个商店里的顾客，在一个码头的船只，一个公司的员工等。离散模拟是指在被模拟时间的某一时刻，进行有待完成的任何事情，然后把模拟时间向前移到某个动作按照调度要出现的下一个时间。

与此相对照，一个“连续模拟”则是在连续变化下动作的模拟，例如在高速公路上车辆的移动，飞向其它星球的宇宙飞船，等等。连续模拟通常可以通过步骤之间的时间间隔非常小的离散模拟满意地近似；然而，在这种情况下，我们通常有“同步”的离散模拟，其中系统的许多部分在每个离散的时间间隔里轻微改变。因此这样一个应用一般要求稍微不同于这里所考虑类型的程序组织类型。

以下所编写的程序模拟在加利福尼亚理工学院的数学楼里的电梯。这样一个模拟的结果也许只对相当经常地访问加利福尼亚理工学院的人有用;而且即使是对于他们,恐怕乘几次电梯就行了,也用不着自找麻烦来写一个计算机程序。但是,如同对于模拟研究通常所做的那样,我们在编写程序中将使用的方法要比由程序给出的答案更有趣得多。以下讨论的方法说明对于离散模拟程序使用的典型的实现技术。

数学楼有五层:底地下室,地下室,一层,二层和三层。有一部电梯,它有自动控制器并且可以在每层处停下。为方便起见,我们把楼层重新编号为0,1,2,3,4。

在每层处,有两个呼叫按钮,一个是供 UP(上)的,一个是供 DOWN(下)的。(实际上,0层只有 UP 而4层只有 DOWN,但我们可以忽略这个异常性,因为多余的按钮将不被使用。)对应于这些按钮,有10个变量 CALLUP[j]和 CALLDOWN[j],$0\leqslant j\leqslant 4$。还有变量 CALLCAR[j],$0\leqslant j\leqslant 4$,表示在电梯内的按钮,它命令电梯开往目的地层。当一个人按按钮时,对应的变量被置为1;在该要求被实现之后电梯把该变量清零。

迄今为止我们已经从用户的观点描述了电梯,但从电梯的角度来看状况更为有趣。电梯处于三个状态之一:GOINGUP(向上),GOINGDOWN(向下)或 NEUTRAL(中性,即不动)。(当前的状态通过电梯内发亮的箭头来向乘客指示。)如果它处于 NEUTRAL 的状态而且不在2层上,电梯门将关闭而且(如果在关门的时刻没有任何命令发出)它将改变成 GOINGUP 或 GOINGDOWN,朝向2层。(这是"大厅层",因为大多数乘客都从这里进入)。在 NEUTRAL 状态下在2层时,电梯最终将关门并静静等候另一个命令。所收到的去向另一层的头一个命令把机器置成相应的 GOINGUP 或 GOINGDOWN;它处于这个状态直到没有在同一个方向等候的命令为止,然后依赖于在 CALL(呼叫)变量中有什么其它的命令,在打开门之前它改变方向或者改变成 NEUTRAL。电梯花费一段时间开门或关门,加速或减速,以及从一层到另外一层。所有这些量都在以下的算法中指出,它比起通俗描述来要更为精确。我们现在将要研究的算法可能不反映电梯操作的真正原理,但可以相信,它提供了一组最简单的规则,足以解释作者在写本小节时,经过好多小时的实验观察到的所有现象。

使用两个共行程序来模拟这个电梯系统,一个是乘客(用户)的,另一个是电梯的;这两个程序确定要实现的所有动作,以及在模拟过程中要使用的各种时间延迟。在下列描述中,变量 TIME 表示被模拟时钟的当前值。所有时间单位以十分之一秒给出。还有若干其它的变量。

FLOOR,电梯当前所处位置;
D1,除当人们进入电梯或离开电梯的时间外,其值为零的一个变量;
D2,如果电梯在某层处不动30秒或以上,其值变为零的一个变量;
D3,除了当电梯门打开但无人进出电梯之外其值为零的一个变量;
STATE,电梯的当前状态(GOINGUP,GOINGDOWN 或 NEUTRAL)。

开始时,FLOOR = 2,D1 = D2 = D3 = 0,且 STATE = NEUTRAL。

共行程序 U(用户) 每一个进入系统的人开始执行以下所确定的动作,并且在步骤 U1 处开始。

U1. [进入,为后继者做准备] 下列的量由将不在这里描述的某种方式所确定:

IN,新用户进入系统时所在的层;

OUT,这个用户要去的层(OUT≠IN);

GIVEUPTIME,在失去耐性决定走楼梯之前这个用户将等候电梯的时间量。

INTERTIME,在另外一个用户将进入系统之前的时间量。

在计算出这些量之后,模拟程序对事情进行设置使得另一个用户在 TIME + INTERTIME 时进入系统。

U2. [发信号和等候] (这一步的目的是传呼电梯;如果电梯已经处于正确的楼层上,便出现一些特殊情况。)如果 FLOOR = IN,而且如果电梯的下一个动作是以下的步骤 E6(即,如果电梯的门正在关闭),则驱使电梯立即转到它的步骤 E3 并且取消它的动作 E6。(这意味着在电梯移动之前它的门将再次打开。)如果 FLOOR = IN 以及如果 D3≠0,则置 D3←0,并把 D1 置为一个非零的值,并且再次启动电梯的活动 E4。(这意味着电梯的门在这个楼层是打开的,但其他人已经进入或离开。电梯步骤 E4 是按照通常的礼仪规则准许人们进入电梯的顺序步骤;因此重新启动 E4 提供给这位用户一个机会在电梯门关闭之前进入电梯。)在所有其它情况下,按照 OUT > IN 或者 OUT < IN,用户设置 CALLUP[IN]←1 或者CALLDOWN[IN]←1;而且如果 D2 = 0 或者电梯处于它的"休眠"位置 E1,就实施下面所描述的 DECISION 子程序。(DECISION 子程序用来在某些关键时刻使电梯脱离 NEUTRAL 状态。)

U3. [进入队列] 把这个用户插入到 QUEUE[IN]的尾部,它是表示在这个楼层等候的人的一个线性表。现在用户耐心等候 GIVEUPTIME 这样长的时间,除非电梯先到达——更精确地说,除非下面的电梯程序步骤 E4 把这个用户送到 U5 并且取消安排好的活动 U4。

U4. [放弃] 如果 FLOOR≠IN 或 D1 = 0,则从 QUEUE[IN]和从模拟系统删除这个用户。(用户觉得,电梯太慢了,或者有一点锻炼要比乘电梯更好些。)如果 FLOOR = IN 以及 D1≠0,用户就停在那里等候(知道等候不会太久)。

U5. [进电梯] 这个用户现在离开 QUEUE[IN]并进入 ELEVATOR,它是表示此刻人在电梯上的一个类似栈的表。置 CALLCAR[OUT]←1。

现在如果 STATE = NEUTRAL,按情况置 STATE←GOINGUP 或 GOINGDOWN,并且在 25 个单位时间之后,置成准备执行电梯活动 E5。(这是电梯的一个特殊特性,当用户选择一个目的楼层时,如果电梯正处于 NEUTRAL 状态,允许门关得比通常要快些。25 个单位时间的间隙给了步骤 E4 一个机会来确保在步骤 E5 时,D1 被适当地建立,关门动作出现。)

现在用户等候直到由下面的步骤 E4 把他送到步骤 U6 为止,这时电梯已经达到所要求楼层。

U6. [走出电梯] 从 ELEVATOR 表和模拟系统删去这个用户。 ▌

共行程序 E(电梯)　这个共行程序表示电梯的动作;其中步骤 E4 也处理当乘客进

入和走出时的控制。

E1. [等候传呼] (这时电梯正在 2 层且门关闭,等候某个事情发生。)如果某人按一个按钮,DECISION 子程序将把我们带到步骤 E3 或 E6 处。眼下是等候。

E2. [状态改变了吗?] 如果 STATE = GOINGUP 且对所有 j > FLOOR, CALLUP[j] = CALLDOWN[j] = CALLCAR[j] = 0,则根据对所有 j < FLOOR,是否 CALLCAR[j] = 0 来设置 STATE←NEUTRAL 或 STATE←GOINGDOWN,并且把当前楼层所有的 CALL 变量置成零;如果 STATE = GOINGDOWN,则以相反的方向做类似动作。

E3. [开门] 把 D1 和 D2 置成任何非零的值。在 300 个单位时间之后,置电梯活动 E9 独立地启动。(这个活动在它出现之前可以在步骤 E6 中被取消。如果它已被安排而且未被取消,我们就取消并且重新安排它。)在 76 个单位时间后,还置电梯活动 E5 独立地启动。然后等候 20 个时间单位(来模拟开门)并转到 E4。

E4. [让乘客进出] 如果在 ELEVATOR 表中的任何人有 OUT = FLOOR,就把最近进入的这类乘客立即送到步骤 U6,等候 25 个时间单位,并重复步骤 E4。如果没有这样的用户存在,但是 QUEUE[FLOOR]不空,则把这队列前端的人立即送到步骤 U5 而不是 U4,等候 25 个单位时间,并且重复步骤 E4。但如果 QUEUE[FLOOR]为空,则置 D1←0,使 D3 非零,并等候某个其它活动来启始进一步的活动。(步骤 E5 将把我们送到 E6,否则步骤 U2 将重新启动 E4。)

E5. [关门] 如果 D1≠0,等候 40 个时间单位并重复这一步骤(门震动一会儿,但它们再弹回打开,因为某个人仍然在进或出)。否则置 D3←0,并且在 25 个时间单位之后置电梯在步骤 E6 处启动。(这模拟在乘客完全进入或离开后的关门;但如果新乘客在这层上进入而门关上,如同在步骤 U2 所指出的那样门将再次打开。)

E6. [准备移动] 把 CALLCAR[FLOOR]置成零;而且如果 STATE≠GOINGDOWN,置 CALLUP[FLOOR]为零,如果 STATE≠GOINGUP,则置 CALLDOWN[FLOOR]为零。(注:如果 STATE = GOINGUP,则电梯不清除 CALLDOWN,因为它假定将要下来的人还未进入;但参见习题 6。)现在实施 DECISION 子程序。

如果在执行 DECISION 子程序之后,STATE = NEUTRAL,则转向 E1。否则,如果 D2≠0,则取消电梯活动 E9。最后,如果 STATE = GOINGUP,等候 15 个时间单位(因为电梯在加速)并转向 E7;如果 STATE = GOINGDOWN,等候 15 个时间单位并转向 E8。

E7. [上一层] 置 FLOOR←FLOOR + 1,并且等候 51 个时间单位。如果现在 CALLCAR[FLOOR] = 1 或 CALLUP[FLOOR] = 1 或者如果((FLOOR = 2 或 CALLDOWN[FLOOR] = 1)而且对于所有 j > FLOOR, CALLUP[j] = CALLDOWN[j] = CALLCAR[j] = 0),等候 14 个时间单位(为了减速)并转向 E2。否则,重复此步。

E8. [下一层] 这一步和 E7 相似只是方向相反,另外把时间 51 和 14 分别改成 61 和 23。(电梯降落比上升要花费更长时间。)

E9. [设置无动作指示符] 置 D2←0 并执行 DECISION 子程序。(这个独立的动作在步骤E3 中启动,但几乎总是在步骤 E6 中取消,参见习题 4。) ▌

表 1　电梯系统的某些动作

TIME	STATE	FLOOR	D1	D2	D3	步骤	动　　作
0000	N	2	0	0	0	U1	用户 1 抵达 0 层,目的地是 2 层
0035	D	2	0	0	0	E8	电梯下降
0038	D	1	0	0	0	U1	用户 2 抵达 4 层,目的地是 1 层
0096	D	1	0	0	0	E8	电梯下降
0136	D	0	0	0	0	U1	用户 3 抵达 2 层,目的地是 1 层
0141	D	0	0	0	0	U1	用户 4 抵达 2 层,目的地是 1 层
0152	D	0	0	0	0	U4	用户 1 决定放弃,离开系统
0180	D	0	0	0	0	E2	电梯停止
0180	N	0	0	X	0	E3	电梯门开始打开
0200	N	0	X	X	0	E4	门打开,但无人在
0256	N	0	0	X	X	E5	电梯门开始关闭
0291	U	0	0	X	0	U1	用户 5 抵达 3 层,目的地是 1 层
0291	U	0	0	X	0	E7	电梯上升
0342	U	1	0	X	0	E7	电梯上升
0364	U	2	0	X	0	U1	用户 6 抵达 2 层,目的地是 1 层
0393	U	2	0	X	0	E7	电梯上升
0444	U	3	0	X	0	E7	电梯上升
0509	U	4	0	X	0	E2	电梯停止
0509	N	4	0	X	0	E3	电梯门开始打开
0529	N	4	X	X	0	U5	用户 2 进入
0540	D	4	X	X	0	U4	用户 6 决定放弃,离开系统
0554	D	4	0	X	X	E5	电梯门开始关闭
0589	D	4	0	X	0	E8	电梯下降
0602	D	3	0	X	0	U1	用户 7 抵达 1 层,目的地是 2 层
0673	D	3	0	X	0	E2	电梯停止
0673	D	3	0	X	0	E3	电梯门开始打开
0693	D	3	X	X	0	U5	用户 5 进入
0749	D	3	0	X	X	E5	电梯门开始关闭
0784	D	3	0	X	0	E8	电梯下降
0827	D	2	0	X	0	U1	用户 8 抵达 1 层,目的地是 0 层
0868	D	2	0	X	0	E2	电梯停止
0868	D	2	0	X	0	E3	电梯门开始打开
0876	D	2	X	X	0	U1	用户 9 抵达 1 层,目的地是 3 层
0888	D	2	X	X	0	U5	用户 3 进入
0913	D	2	X	X	0	U5	用户 4 进入
0944	D	2	0	X	X	E5	电梯门开始关闭
0979	D	2	0	X	0	E8	电梯下降
1048	D	1	0	X	0	U1	用户 10 抵达 0 层,目的地是 4 层
1063	D	1	0	X	0	E2	电梯停止
1063	D	1	0	X	0	E3	电梯门开始打开

（续）

TIME	STATE	FLOOR	D1	D2	D3	步骤	动 作
1083	D	1	X	X	0	U6	用户 4 走出电梯，离开系统
1108	D	1	X	X	0	U6	用户 3 走出电梯，离开系统
1133	D	1	X	X	0	U6	用户 5 走出电梯，离开系统
1139	D	1	X	X	0	E5	门震动
1158	D	1	X	X	0	U6	用户 2 走出电梯，离开系统
1179	D	1	X	X	0	E5	门震动
1183	D	1	X	X	0	U5	用户 7 进入
1208	D	1	X	X	0	U5	用户 8 进入
1219	D	1	X	X	0	E5	门震动
1233	D	1	X	X	0	U5	用户 9 进入
1259	D	1	0	X	X	E5	电梯门开始关闭
1294	D	1	0	X	0	E8	电梯下降
1378	D	0	0	X	0	E2	电梯停止
1378	U	0	0	X	0	E3	电梯门开始打开
1398	U	0	X	X	0	U6	用户 8 走出电梯，离开系统
1423	U	0	X	X	0	U5	用户 10 进入
1454	U	0	0	X	X	E5	电梯门开始关闭
1489	U	0	0	X	0	E7	电梯上升
1554	U	1	0	X	0	E2	电梯停止
1554	U	1	0	X	0	E3	电梯门开始打开
1630	U	1	0	X	X	E5	电梯门开始关闭
1665	U	1	0	X	0	E7	电梯上升
……							
4257	N	2	0	X	0	E1	电梯休眠
4384	N	2	0	X	0	U1	用户 17 抵达 2 层，目的地是 3 层
4404	N	2	0	X	0	E3	电梯门开始打开
4424	N	2	X	X	0	U5	用户 17 进入
4449	U	2	0	X	X	E5	电梯门开始关闭
4484	U	2	0	X	0	E7	电梯上升
4549	U	3	0	X	0	E2	电梯停止
4549	N	3	0	X	0	E3	电梯门开始打开
4569	N	3	X	X	0	U6	用户 17 走出电梯，离开系统
4625	N	3	0	X	X	E5	电梯门开始关闭
4660	D	3	0	X	0	E8	电梯下降
4744	D	2	0	X	0	E2	电梯停止
4744	N	2	0	X	0	E3	电梯门开始打开
4764	N	2	X	X	0	E4	门打开，但无人在
4820	N	2	0	X	0	E5	电梯门开始关闭
4840	N	2	0	X	0	E1	电梯休眠
……							

子程序 D(DECISION 子程序)　如同上面的共行程序所描述那样,这个子程序在某些关键时刻被执行,即当需要对电梯的下一方向作出决策时。

D1. [决策必要吗?] 如果 STATE≠NEUTRAL,从本子程序离开。

D2. [门应当打开吗?] 如果电梯正处于 E1 的位置而且如果 CALLUP[2],CALLCAR[2]和 CALLDOWN[2]不全为零,则使电梯在 20 个时间单位之后启动它的动作 E3,且从这个子程序离开。(如果 DECISION 子程序当前正被独立的活动 E9 调用,电梯共行程序有可能处于 E1 的位置处。)

D3. [有任何呼叫吗?] 找出使 CALLUP[j],CALLCAR[j]或 CALLDOWN[j]非零的最小 $j\neq$ FLOOR,并前进到步骤 D4。但如果不存在这样的 j,若 DECISION 当前正由步骤 E6 所调用,则置 $j\leftarrow 2$;否则从这个子程序离开。

D4. [置 STATE] 如果 FLOOR > j,则置 STATE←GOINGDOWN;如果 FLOOR < j,则置 STATE←GOINGUP。

D5. [电梯休眠吗?] 如果电梯共行程序正处于步骤 E1 处,且 $j\neq 2$,则在 20 个时间单位后电梯执行步骤 E6。从这个子程序离开。 ▍

同本书我们已经见过的其它算法相比,上面描述的电梯系统是十分复杂的,但对现实生活中系统的这个选择比起任何虚构的"教科书例子"来是更为典型的模拟问题。

为了帮助理解这个系统,考虑表 1,它给出了一个模拟的部分历史。或许最好是考察从时间 4257 开始的简单情况:当一个用户抵达时(时间为 4384),电梯正在第 2 层处于空闲状态,而且它的门关闭;假设用户的名字是唐。2 秒钟后,门打开,再过 2 秒,唐进入电梯。通过按按钮"3"他启动电梯往上;最终他在 3 层离开,而电梯回到第 2 层。

表 1 前面的项示出多得多的戏剧性场面:一个用户要求电梯到 0 层,但在 15.2 秒之后他失去耐性而放弃了。电梯停在 0 层但那里没有人;然后它上到第 4 层,因为有好多个传呼要下楼;等等。

对一台计算机(在我们的情况下,即 MIX)进行该系统的程序设计值得仔细研究。在模拟期间的任何时刻,系统中都可能有许多被模拟的用户(在各种队列中以及在各种时刻已经做好"放弃"准备的),而当电梯正在试图关门时如果许多人打算离开,也有实质上同时执行 E4,E5 和 E9 的可能。被模拟时间的消逝以及"同时性"的处理可以通过由一个节点表示的每个实体编程,这节点包括一个 NEXTTIME 字段(表示对于这个实体下一个动作要发生的时间)以及一个 NEXTINST 字段(表示这个实体开始执行指令的内存地址,这类似于通常的共行程序链接)。等候时间消逝的每一个实体被放置在称做 WAIT 的一个双重链接表中;这个"日程"按它的节点的 NEXTTIME 字段排序,使得这些动作可以以正确的模拟时间序列来加以处理。对于 ELEVATOR 和对于 QUEUE 表,这个程序也使用双重链接表。

表示一个活动(无论是用户活动还是电梯活动)的每个节点有下列形式:

+	IN	LLINK1	RLINK1	
+	NEXTTIME			
+	NEXTINST	0	0	39
+	OUT	LLINK2	RLINK2	

(6)

这里 LLINK1 和 RLINK1 是 WAIT 表的链接;LLINK2 和 RLINK2 用做在 QUEUE 表或 ELEVATOR 中的链接。后两个字段和 IN 及 OUT 字段,当节点(6)表示一个用户时,是相关的,但对于表示电梯动作的节点是不相关的。这个节点的第三个字实际上是一个 MIX 的“JMP”(转移)指令。

图 12 示出 WAIT 表,ELEVATOR 表以及一个 QUEUE 的典型内容;在 QUEUE 表中的每个节点同时也在 WAIT 表中但有 NEXTINST = U4,不过这在图中未指出,因为如果把链接复杂化将使基本思想反而模糊。

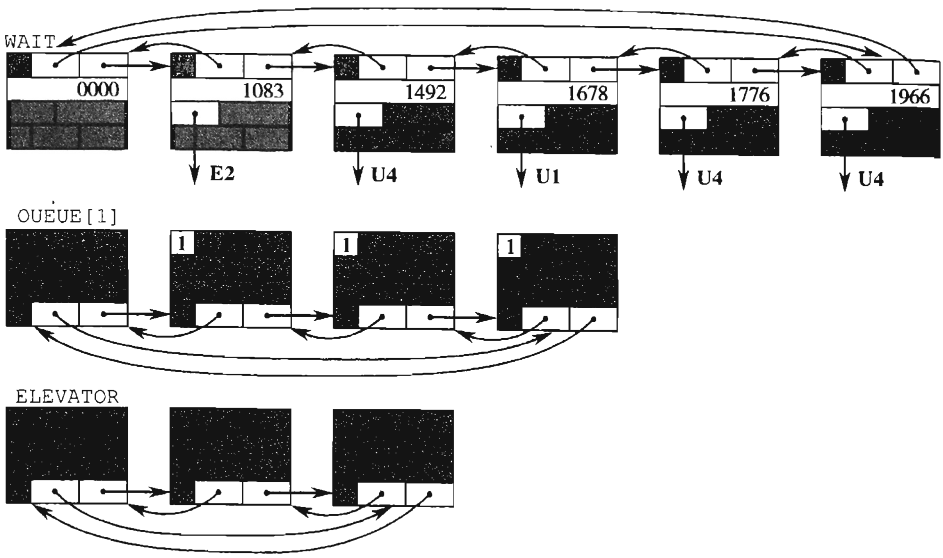


图 12 用于电梯模拟程序中的某些表(表头出现在左边)

现在考虑程序本身。它十分长,尽管(如同所有的长程序那样)它分成一些小的部分,而每个部分本身十分简单。首先是一些只用来定义表格的初始内容的代码行。在这里有若干有趣之点,我们有对于 WAIT 表(行 010 ~ 011),QUEUE 表(行 026 ~ 031)和 ELEVATOR 表(行 032 ~ 033)的表头。它们的每一个都是形如(6)的节点,但把一些不重要的字删去。WAIT 表头仅包含一个节点的头两个字;而 QUEUE 和 ELEVATOR 表头仅要求一个节点的最后一个字。还有总是在系统中出现的四个节点(行 012 ~ 023):USER1,总是在步骤 U1 处放置的准备好把一个新用户输入到系统中的节点;ELEV1,在步骤 E1,

E2,E3,E4,E6,E7 和 E8 支配电梯的主要动作的节点;以及 ELEV2 和 ELEV3,为电梯动作 E5 和 E9 所使用的节点,它们相对于模拟时间而言独立于其它的电梯动作而发生。这四个节点的每一个都只含三个字,因为它们绝不出现在 QUEUE 或 ELEVATOR 表中。表示系统中每个实际用户的节点将出现在主程序后面的存储池中。

```
 * THE ELEVATOR SIMULATION
IN          EQU  1:1                                  诸节点内诸字段的定义
LLINK1      EQU  2:3
RLINK1      EQU  4:5
NEXTINST    EQU  0:2
OUT         EQU  1:1
LLINK2      EQU  2:3
RLINK2      EQU  4:5

 *  FIXED-SIZE TABLES AND LIST HEADS
WAIT        CON  *+2(LLINK1),*+2(RLINK1)             WAIT 表的表头
            CON  0                                      总是有 NEXTTIME=0
USER1       CON  *-2(LLINK1),*-2(RLINK1)             此节点表示动作 U1
            CON  0                                      且开始时它是
            JMP  U1                                     WAIT 表的惟一入口
ELEV1       CON  0                                    此节点表示电梯动作,
            CON  0                                      但 E5 和 E9 除外
            JMP  E1
ELEV2       CON  0                                    此节点表示 E5 处的
            CON  0                                      独立的电梯动作
            JMP  E5
ELEV3       CON  0                                    此节点表示 E9 处的
            CON  0                                      独立的电梯动作
            JMP  E9
AVAIL       CON  0                                    链接到可用节点
TIME        CON  0                                    当前模拟时间
QUEUE       EQU  *-3
            CON  *-3(LLINK2),*-3(RLINK2)             QUEUE[0]的表头
            CON  *-3(LLINK2),*-3(RLINK2)             QUEUE[1]的表头
            CON  *-3(LLINK2),*-3(RLINK2)             所有队列开始时
            CON  *-3(LLINK2),*-3(RLINK2)               为空
            CON  *-3(LLINK2),*-3(RLINK2)             QUEUE[4]的表头
ELEVATOR    EQU  *-3
            CON  *-3(LLINK2),*-3(RLINK2)             ELEVATOR 的表头
```

```
034              CON  0                                   ⎫ CALL 表的“补空”
035              CON  0                                   ⎬ (见行 183 ~ 186)
036              CON  0                                   ⎪
037              CON  0                                   ⎭
038  CALL        CON  0          CALLUP[0],CALLCAR[0],CALLDOWN[0]
039              CON  0          CALLUP[1],CALLCAR[1],CALLDOWN[1]
040              CON  0          CALLUP[2],CALLCAR[2],CALLDOWN[2]
041              CON  0          CALLUP[3],CALLCAR[3],CALLDOWN[3]
042              CON  0          CALLUP[4],CALLCAR[4],CALLDOWN[4]
043              CON  0                                   ⎫ CALL 表的“补空”
044              CON  0                                   ⎬ (见行 178 ~ 181)
045              CON  0                                   ⎪
046              CON  0                                   ⎭
047  D1          CON  0                                   表示门打开,活动
048  D2          CON  0                                   表示非长时间的静止
049  D3          CON  0                                   表示门打开,不活动 ▌
```

程序代码的下一部分包含模拟过程的基本子程序和主要控制程序。子程序 INSERT 和 DELETE 实现对双重链接表的典型操作;它们把当前的节点放进一个 QUEUE 或 ELEVATOR 表或者从 QUEUE 或 ELEVATOR 表取出一个节点。(在这个程序中,“当前节点”C 总是通过变址寄存器 6 来表示。)还有处理 WAIT 表的子程序:子程序 SORTIN 把当前节点加到 WAIT 表中,并且基于它的 NEXTTIME 字段把它排序到正确的位置。子程序 IMMED 把当前的节点插在 WAIT 表的前端。子程序 HOLD 把当前的节点插入 WAIT 表,且 NEXTTIME 等于当前的时间加上寄存器 A 中的数量。子程序 DELETEW 从 WAIT 表删去当前的节点。

程序 CYCLE 是模拟控制的心脏:它判定下一步执行什么动作(即 WAIT 表的头一个元素,我们知道 WAIT 表非空),并且转移到相应入口。CYCLE 有两个特殊的入口:CYCLE1 首先把 NEXTINST 置成当前节点;HOLDC 做同样操作,但另外加上对 HOLD 子程序的调用。因此若在寄存器 A 中有数量 t,则指令“JMP HOLDC”的效果是搁置活动 t 个模拟时间单位,而后返回到下一个单元。

```
050  * SUBROUTINES AND CONTROL ROUTINE
051  INSERT    STJ   9F             插入 NODE(C)到 NODE(rI1)左边
052            LD2   3,1(LLINK2)    rI2←LLINK2(rI1)
053            ST2   3,6(LLINK2)    LLINK2(C)←rI2
054            ST6   3,1(LLINK2)    LLINK2(rI1)←C
055            ST6   3,2(RLINK2)    RLINK2(rI2)←C
056            ST1   3,6(RLINK2)    RLINK2(C)←rI1
057  9H        JMP   *              从子程序离开
```

```
058  DELETE   STJ   9F             从其表中删去 NODE(C)
059           LD1   3,6(LLINK2)    P←LLINK2(C)
060           LD2   3,6(RLINK2)    Q←RLINK2(C)
061           ST1   3,2(LLINK2)    LLINK2(Q)←P
062           ST2   3,1(RLINK2)    RLINK2(P)←Q
063  9H       JMP   *              从子程序离开
064  IMMED    STJ   9F             把 NODE(C)插入 WAIT 表前端:
065           LDA   TIME
066           STA   1,6            置 NEXTTIME(C)←TIME
067           ENT1  WAIT           P← LOC(WAIT)
068           JMP   2F             把 NODE(C)插入 NODE(P)右边
069  HOLD     ADD   TIME           rA←TIME + rA
070  SORTIN   STJ   9F             把 NODE(C)排序于 WAIT 表中
071           STA   1,6            置 NEXTTIME(C)←rA
072           ENT1  WAIT           P←LOC(WAIT)
073           LD1   0,1(LLINK1)    P←LLINK1(P)
074           CMPA  1,1            从右到左比较 NEXTTIME 字段
075           JL    * - 2          重复直至 NEXTTIME(C)≥NEXTTIME(P)
076  2H       LD2   0,1(RLINK1)    Q←RLINK1(P)
077           ST2   0,6(RLINK1)    RLINK1(C)←Q
078           ST1   0,6(LLINK1)    LLINK1(C)←P
079           ST6   0,1(RLINK1)    RLINK1(P)←C
080           ST6   0,2(LLINK1)    LLINK1(Q)←C
081  9H       JMP   *              从子程序离开
082  DELETEW  STJ   9F             从 WAIT 表删去 NODE(C):
083           LD1   0,6(LLINK1)    (这和行 058 ~ 063 一样,
084           LD2   0,6(RLINK1)      除了用的是 LLINK1 和 RLINK1
085           ST1   0,2(LLINK1)      而不是 LLINK2,RLINK2 外)
086           ST2   0,1(RLINK1)
087  9H       JMP   *
088  CYCLE1   STJ   2,6(NEXTINST)  置 NEXTINST(C)←rJ
089           JMP   CYCLE
090  HOLDC    STJ   2,6(NEXTINST)  置 NEXTINST(C)←rJ。
091           JMP   HOLD           把 NODE(C)插入 WAIT,延迟 rA。
092  CYCLE    LD6   WAIT(RLINK1)   置当前节点 C←RLINK1(LOC(WAIT))。
093           LDA   1,6
094           STA   TIME           TIME←NEXTTIME(C)
095           JMP   DELETEW        从 WAIT 表删去 NODE(C)
```

```
096            JMP   2,6             转到 NEXTINST(C) ∎
```

现在讨论共行程序 U 的程序。在步骤 U1 的开始，当前节点 C 是 USER1（见上面的行 012~014)，而且程序的 099~100 行引起 USER1 被重新插入 WAIT 表使得在 INTERTIME 个模拟时间单位之后将生成下一个用户。以下的行 101~114 负责对新近生成的用户建立一个节点，IN 和 OUT 的层号被记录在这个节点位置中。AVAIL 栈在每个节点的 RLINK1 中被单个地链接。注意行 101~108 利用 POOLMAX 技术 2.2.3-(7)执行动作"C⇐AVAIL"；在这里不需要进行对 OVERFLOW 的测试，因为存储池总的大小（在任意一个时间下系统中的用户数）很少超过 10 个节点（40 个字）。把一个节点返回到 AVAIL 栈出现于行 156~158 中。

贯穿这个程序，变址寄存器 4 等于变量 FLOOR，而且依赖于是否 STATE = GOINGUP，GOINGDOWN或 NEUTRAL，变址寄存器 5 为正、负或零。变量 CALLUP[*j*]，CALLCAR[*j*]和 CALLDOWN[*j*]分别占有 CALL + *j* 单元的(1:1)，(3:3)以及(5:5)字段。

```
097   * COROUTINE U                          U1.进入，为后继者做准备
098   U1       JMP   VALUES                  置 INFLOOR,OUTFLOOR,GIVEUPTIME,INTERTIME
099            LDA   INTERTIME               INTERTIME 由 VALUES 子程序计算
100            JMP   HOLD                    把 NODE(C)放入 WAIT,延迟 INTERTIME
101            LD6   AVAIL                   C←AVAIL
102            J6P   1F                      如果 AVAIL≠Λ,则转移
103            LD6   POOLMAX(0:2)
104            INC6  4                       C←POOLMAX +4
105            ST6   POOLMAX(0:2)            POOLMAX←C
106            JMP   * +3                    假定内存溢出不发生
107   1H       LDA   0,6(RLINK1)
108            STA   AVAIL                   AVAIL←RLINK1(AVAIL)
109            LD1   INFLOOR                 rI1←INFLOOR(由上面的 VALUES 计算)
110            ST1   0,6(IN)                 IN(C)←rI1
111            LD2   OUTFLOOR                rI2←OUTFLOOR(由 VALUES 计算)
112            ST2   3,6(OUT)                OUT(C)←rI2
113            ENTA  39                      把常数 39(JMP 的操作码)放进
114            STA   2,6                       节点格式(6)的第三个字中
115   U2       ENTA  0,4                     U2.发信号并等候。置 rA←FLOOR
116            DECA  0,1                     FLOOR - IN
117            ST6   TEMP                    保存 C 的值
118            JANZ  2F                      如果 FLOOR≠IN,则转移
119            ENT6  ELEV1                   置 C←LOC(ELEV1)
120            LDA   2,6(NEXTINST)           电梯处于 E6 的位置吗?
121            DECA  E6
```

```
           JANZ   3F
           ENTA   E3              如果是,把它重新放置在 E3 处
           STA    2,6(NEXTINST)
           JMP    DELETEW         把它从 WAIT 表删去,并
           JMP    4F                重新插入 WAIT 的前端
     3H    LDA    D3
           JAZ    2F              如果 D3 = 0,则转移
           ST6    D1              否则使 D1 成为非零
           STZ    D3              置 D3←0
     4H    JMP    IMMED           把 ELEV1 插入 WAIT 表前端
           JMP    U3              (rI1 和 rI2 已经改变)
     2H    DEC2   0,1             rI2←OUT - IN
           ENTA   1
           J2P    *+3             如果向上,则转移
           STA    CALL,1(5:5)     置 CALLDOWN[IN]←1
           JMP    *+2
           STA    CALL,1(1:1)     置 CALLUP[IN]←1
           LDA    D2
           JAZ    *+3             如果 D2 = 0,调用 DECISION 子程序
           LDA    ELEV1+2(NEXTINST)
           DECA   E1              如果电梯处于 E1 处,调用
           JAZ    DECISION          DECISION 子程序
     U3    LD6    TEMP            U3. 进入队列
           LD1    0,6(IN)
           ENT1   QUEUE,1         rI1←LOC(QUEUE[IN])
           JMP    INSERT          把 NODE(C)插入 QUEUE[IN]右端
     U4A   LDA    GIVEUPTIME
           JMP    HOLDC           等候 GIVEUPTIME 个时间单元
     U4    LDA    0,6(IN)         U4. 放弃
           DECA   0,4             IN(C) - FLOOR
           JANZ   *+3
           LDA    D1              FLOOR = IN(C)
           JANZ   U4A             参见习题 7
     U6    JMP    DELETE          U6. 离开。从 QUEUE 或 ELEVATOR 删去节
           LDA    AVAIL             点 NODE(C)
           STA    0,6(RLINK1)     AVAIL⇐C
           ST6    AVAIL
           JMP    CYCLE           继续模拟
```

```
160  U5     JMP   DELETE         U5. 进入。从 QUEUE 删去 NODE(C)
161         ENT1  ELEVATOR
162         JMP   INSERT         把它插入到 ELEVATOR 右边
163         ENTA  1
164         LD2   3,6(OUT)
165         STA   CALL,2(3:3)    置 CALLCAR[OUT(C)]←1
166         J5NZ  CYCLE          如果 STATE≠NEUTRAL 则转移
167         DEC2  0,4            rI2←OUT(C) - FLOOR
168         ENT5  0,2            把 STATE 置成适当方向
169         ENT6  ELEV2          置 C←LOC(ELEV2)
170         JMP   DELETEW        从 WAIT 表删去 E5 动作
171         ENTA  25
172         JMP   E5A            从现在开始 25 个时间单位后重新启动 E5
                                   动作   ▌
```

共行程序 E 的程序是较早前给出的半正式描述的相当直截了当的翻译。或许最有趣的部分是在步骤 E3 中为电梯独立动作所做的准备，以及在步骤 E4 中对 ELEVATOR 和 QUEUE 表的查找。

```
173  * COROUTINE E
174  E1A    JMP   CYCLE1         置 NEXTINST←E1，转到 CYCLE
175  E1     EQU   *              E1. 等候呼叫(无动作)
176  E2A    JMP   HOLDC
177  E2     J5N   1F             E2. 状态改变了?
178         LDA   CALL+1,4       状态是 GOINGUP
179         ADD   CALL+2,4
180         ADD   CALL+3,4
181         ADD   CALL+4,4
182         JAP   E3             有对较高楼层的呼叫吗?
183         LDA   CALL-1,4(3:3)  若无，有无电梯中用户对较低楼层的呼叫?
184         ADD   CALL-2,4(3:3)
185         ADD   CALL-3,4(3:3)
186         ADD   CALL-4,4(3:3)
187         JMP   2F
188  1H     LDA   CALL-1,4       状态是 GOINGDOWN
189         ADD   CALL-2,4       动作和行 178 ~ 186 类似
 ⋮
196         ADD   CALL+4,4(3:3)
197  2H     ENN5  0,5            颠倒 STATE 的方向
```

```
        STZ     CALL,4          置 CALL 变量为零
        JANZ    E3              如果呼叫的是相反的方向,则转移;
        ENT5    0                 否则置 STATE←NEUTRAL
E3      ENT6    ELEV3           E3. 开门
        LDA     0,6             如果活动 E9 已被安排,则
        JANZ    DELETEW           从 WAIT 表删去它
        ENTA    300
        JMP     HOLD            在 300 个时间单位之后安排活动 E9
        ENT6    ELEV2
        ENTA    76
        JMP     HOLD            在 76 个时间单位之后安排活动 E5
        ST6     D2              置 D2 为非零
        ST6     D1              置 D1 为非零
        ENTA    20
E4A     ENT6    ELEV1
        JMP     HOLDC
E4      ENTA    0,4             E4. 让用户出入电梯
        SLA     4               把 rA 的 OUT 字段置为 FLOOR
        ENT6    ELEVATOR        C←LOC(ELEVATOR)
1H      LD6     3,6(LLINK2)     C←LLINK2(C)
        CMP6    =ELEVATOR=      从右到左,查找 ELEVATOR 表
        JE      1F              如果 C = LOC(ELEVATOR),查找完成
        CMPA    3,6(OUT)        OUT(C)同 FLOOR 作比较
        JNE     1B              如果不相等,继续查找;
        ENTA    U6                否则准备把用户发送到 U6
        JMP     2F
1H      LD6     QUEUE+3,4(RLINK2)  置 C←RLINK2(LOC(QUEUE[FLOOR]))
        CMP6    3,6(RLINK2)     C = RLINK2(C)吗?
        JE      1F              如果是,则队列为空
        JMP     DELETEW         如果否,对此用户取消动作 U4
        ENTA    U5              准备以 U5 代替 U4
2H      STA     2,6(NEXTINST)   置 NEXTINST(C)
        JMP     IMMED           把用户放在 WAIT 表的前端
        ENTA    25
        JMP     E4A             等候 25 个时间单位并重复 E4
1H      STZ     D1              置 D1←0
        ST6     D3              置 D3 为非零
        JMP     CYCLE           返回来模拟其它事件
```

```
236  E5A   JMP    HOLDC
237  E5    LDA    D1              E5. 关门
238        JAZ    *+3             D1=0 吗?
239        ENTA   40              如果不是,用户仍然在进出电梯
240        JMP    E5A             等候 40 个时间单位,重复 E5
241        STZ    D3              如果 D1=0,置 D3←0
242        ENT6   ELEV1
243        ENTA   20
244        JMP    HOLDC           等候 20 个时间单位,然后转到 E6
245  E6    J5N    *+2             E6. 准备移动
246        STZ    CALL,4(1:3)     如果 STATE≠GOINDOWN,在这一层
247        J5P    *+2               上 CALLUP 和 CALLCAR 被恢复
248        STZ    CALL,4(3:5)     若≠GOINGUP,恢复 CALLCAR 和 CALLDOWN
249        J5Z    DECISION        执行 DECISION 子程序
250  E6B   J5Z    E1A             如果 STATE=NEUTRAL,转到 E1 和等候
251        LDA    D2
252        JAZ    *+4
253        ENT6   ELEV3           否则,如果 D2≠0
254        JMP    DELETEW           取消活动 E9
255        STZ    ELEV3             (见行 202)
256        ENT6   ELEV1
257        ENTA   15              等候 15 个时间单位
258        J5N    E8A             如果 STATE=GOINGDOWN,转到 E8
259  E7A   JMP    HOLDC
260  E7    INC4   1               E7. 向上一层
261        ENTA   51
262        JMP    HOLDC           等候 51 个时间单位
263        LDA    CALL,4(1:3)     是 CALLCAR[FLOOR]还是 CALLUP[FLOOR]
264        JAP    1F              ≠0?
265        ENT1   -2,4            如果不是,
266        J1Z    2F                是否 FLOOR=2?
267        LDA    CALL,4(5:5)     如果不是,是否 CALLDOWN[FLOOR]≠0?
268        JAZ    E7              如果不是,重复步骤 E7
269  2H    LDA    CALL+1,4
270        ADD    CALL+2,4
271        ADD    CALL+3,4
272        ADD    CALL+4,4
273        JANZ   E7              有没有上更高楼层的传呼?
```

```
274  1H       ENTA  14          是停止电梯的时候了
275           JMP   E2A         等候 14 个时间单位并转到 E2
276  E8A      JMP   HOLDC
 ⋮                              (见习题 8)
292           JMP   E2A
293  E9       STZ   0,6         E9. 置不动指示器(见行 202)
294           STZ   D2          D2←0
295           JMP   DECISION    执行 DECISION 子程序
296           JMP   CYCLE       返回以模拟其它事件 ▌
```

在这里我们不考虑 DECISION 子程序(见习题 9),也不考虑用来确定关于电梯的要求的 VALUES 子程序。在程序的末端,是下列代码:

```
BEGIN     ENT4  2          由 FLOOR = 2
          ENT5  0            和 STATE = NEUTRAL 开始
          JMP   CYCLE      开始模拟
POOLMAX   NOP   POOL
POOL      END   BEGIN      存储池在文字和临时存储单元之后 ▌
```

随着上面程序以它自己的步调前进,它确实能很好地模拟电梯系统的的工作;但是运行这个程序将是无用的,因为没有输出！实际上,作者加了一个 PRINT 子程序,它在上面程序的大多数关键步骤被调用,并且用来编制表格 1;其细节已被省略,因为它们是非常直截了当的,而且加进来只会使代码更加杂乱。

已经出现若干种程序设计语言,它们使得描述一个离散系统中的动作变得十分容易,也使用一个编译程序把这些描述翻译成为机器语言十分容易。当然,在这节中使用的是汇编语言,因为在这里我们关心的是链接表操作的基本技术,而且我们要想看看离散模拟如何可以真正地通过有单向思维的计算机加以实现的细节。使用一个 WAIT 表或日程表控制共行程序的顺序,如同我们在这节中所已经做的这样,这样的技术称为**准并行处理**。

对于这样一个长程序的运行时间给出精确的分析是很困难的,因为涉及了复杂的交互作用。但是大的程序通常都把它们的大部分时间花费在相对短的程序上做相对简单的事情。因此通过使用一个叫做记录程序(profiler)的跟踪程序,我们通常可以得到总体效率的较好表征,记录程序执行程序并且记录每条指令被执行的频繁程度。它可以标识出"瓶颈",即应予以特殊注意的位置。[参见习题 1.4.3.2-7。也可参见*Software Practice & Experience* **1** (1971),105 ~ 133,那里对随机选择的 FORTRAN 程序进行了这种研究,那些 FORTRAN 程序是在斯坦福计算中心的垃圾筐中找到的。]作者对上面的电梯程序做了这样一个实验,对它运行了 10000 个模拟时间单位;26 个用户进入到模拟系统中。在 SORTIN 循环中的指令,即行 073 ~ 075,是最经常地被执行的,共 1432 次,而 SORTIN 子程序本身被调用 437 次。CYCLE 程序被执行 407 次;所以通过不调用在 095 行的 DELETEW 子程序,我们可以赢得一点速度:该程序的四行可以全部写到外边来(以

便每次使用 CYCLE 时节省 $4u$ 的时间)。记录程序也表明 DECISION 子程序只被调用 32 次,而在 E4 中的循环(行 217~219)仅被执行 142 次。

如同在编写上例时作者学习电梯的知识那样,希望读者也能从上例中学习尽量多有关模拟的知识。

习　题

1. [*21*] 对(1)中所表示的双重链接表给出在其左端插入和删去信息的说明。(通过对称性可得到在右端的对偶操作,由此我们便有对一般双端队列的所有动作。)

▶ 2. [*22*] 说明为什么单链接的一个表不能像一般的双端队列那样有效地操作;项的删去只能在单链接的表的一端有效地进行。

▶ 3. [*22*] 正文中描述的电梯系统对于每个楼层使用 CALLUP,CALLCAR 以及 CALLDOWN 三个变量,表示已经由系统的用户按下的按钮。可以想像,每个楼层的呼叫按钮实际上只需要一个或两个二进制变量,而不是三个。说明对于这个电梯系统,一个实验者如何能以某种顺序按按钮来证明(除开顶层和底层之外),每层有三个独立的二进制变量。

4. [*24*] 电梯共行程序中的活动 E9 通常被步骤 E6 删去;而且即使不被删去,它也不做太多的事。说明在什么情况下,如果真的从系统中删去活动 E9,则电梯将有不同的动作。例如,它有时将以不同的顺序访问楼层吗?

5. [*20*] 在表 1 中,用户 10 在时间 1048 时抵达楼层 0。说明如果用户 10 已经抵达楼层 2,而不是楼层 0,电梯在接纳了楼层 1 的用户之后,将向上而不是向下,尽管用户 8 要下到 0 层去。

6. [*23*] 在表 1 中的时间 1183~1233 里,用户 7,8 和 9 都在层 1 进入电梯。然后电梯下到 0 层且只有用户 8 离开电梯。现在电梯再次在 1 层处停下,假想地接已经在电梯上的用户 7 和 9;实际上在楼层 1 处没有任何人等候上电梯。(这种情况在加州理工学院经常出现;如果你已在向着错误方向行进的电梯上,你必须等候一个额外的停止,就如同你再次由原来的楼层上电梯那样。)在许多电梯系统中,用户 7 和 9 将不在时间 1183 时上电梯,因为电梯外面的指示灯将显示它在往下而不是往上;那些用户将等候电梯向上回来并停止下来接他们。在所描述的系统上,没有这样的指示灯,因此不可能在你进电梯之前告知它是否往上;因此表 1 就反映了这个实际情况。

如果我们要来模拟相同的电梯系统,但它带有指示灯,则对共行程序 U 和 E 应作什么改变,使得当电梯的状态是同人们所希望去的方向相反时,他们不上电梯?

7. [*25*] 尽管程序中的错误经常使程序员感到难堪,但如果我们要从错误中学习,就应该把它们记录下来并且告诉给其他人而不是把它们忘掉。(除了其它错误外)以下的错误是当作者头一次写本小节的程序时所犯的错误:行 154 是“JANZ CYCLE”而不是“JANZ U4A”。其原因是如果电梯确实已经抵达了用户所在楼层,那就没有任何理由还去执行“放弃”活动,所以我们可以只转去 CYCLE 并继续模拟其它的活动。错在哪里?

8. [*21*] 写出对于步骤 E8,即行 277~292 的代码,在正文中的程序里它们被省略了。

9. [*23*] 写出 DECISION 子程序的代码,在正文中的程序里它们被省略了。

10. [*40*] 或许指出这样一点是重要的,即尽管作者已经使用电梯系统许多年,而且他自以为已很了解它了,但在他试图写本小节的电梯程序之前,关于选择电梯系统的方向方面还存在有不少事实,他还未认识到。作者前去做电梯实验共六次,每次都以为最后实现了对电梯操纵法的

彻底了解。(现在作者不大愿意再去乘电梯了,因为害怕将出现关于电梯操作的新事实,同这里给出的算法相悖。)我们经常在试图用计算机模拟某一事物之前,都没有认识到,关于这件事物我们所知甚微。

试给出你熟悉的某个电梯的动作的描述。通过对该电梯本身的实验检查你的算法(考察它的线路是不公平的!);然后设计对于该系统的一个离散的模拟程序,并在一台计算机上运行它。

▶ 11. [*21*] (内容变化很少的内存) 以下的问题经常出现在同步模拟中:系统有 n 个变量 V[1],…,V[n],而且在每个模拟步骤中,由它们的旧值来计算它们中某一些的新值。在所有赋值都已完成之前,这些变量不会改变成它们的新值。在这个意义下,假定这些计算是"同时"进行的。因此,出现在相同模拟时间的两个语句

$$\text{V}[1]\leftarrow\text{V}[2] \text{ 和 } \text{V}[2]\leftarrow\text{V}[1]$$

将交换 V[1]和 V[2]的值;这同在顺序计算中发生的情况十分不同。

所希望的动作当然可以通过保持一张附加的表 NEWV[1],…,NEWV[k],$1\le k\le n$ 来进行模拟。在每个模拟步骤之前,对于 $1\le k\le n$,我们可以置 NEWV[k]←V[k],然后记录在 NEWV[k]中所有 V[k]的改动,而且最后,在这个步骤之后,我们可以对 $1\le k\le n$ 设置 V[k]←NEWV[k]。但由于下列原因这个"硬算"方法不完全令人满意:(1) n 经常很大,在每一步中被改变的变量的个数比较小;(2) 这些变量通常不是安排在一个很好的表格 V[1],…,V[n]中,而是以相当混乱的方式散布在整个内存当中;(3) 当在同一个模拟步骤中对一个变量给了两个值时,这个方法探测不出这种状况(通常在这个模型中这是一个错误)。

假定在每步中被改变的变量的数目比较小,试设计一个模拟所希望动作的有效算法,并且使用两个辅助表 NEWV[k]和 LINK[k],$1\le k\le n$。如果可能,在同一步骤中同一个变量被给予两个不同的值时,你的算法应给出一个出错暂停。

▶ 12. [*22*] 本小节的模拟程序中使用了一个双重链接表,而未使用单链接表或顺序表,为什么这是一个好的想法?

2.2.6 数组和正交表

线性表的最简单推广之一,是信息的二维或更高维的数组。例如,考虑一个 $m\times n$ 矩阵

$$\begin{pmatrix} \text{A}[1,1] & \text{A}[1,2] & \cdots & \text{A}[1,n] \\ \text{A}[2,1] & \text{A}[2,2] & \cdots & \text{A}[2,n] \\ \vdots & \vdots & & \vdots \\ \text{A}[m,1] & \text{A}[m,2] & \cdots & \text{A}[m,n] \end{pmatrix} \tag{1}$$

的情况。在这个二维数组中,每个节点 A[j,k]属于两个线性表:"行 j"表 A[j,1],A[j,2],…,A[j,n]和"列 k"表 A[1,k],A[2,k],…,A[m,k]。这些正交的行和列表实质上解释一个矩阵的二维结构。类似的说明也适用于更高维的信息数组。

顺序分配　当把一个类似(1)的数组储存于顺序的内存单元时,存储器通常被分配成使得

$$\text{LOC}(\text{A}[\text{J},\text{K}]) = a_0 + a_1\text{J} + a_2\text{K} \tag{2}$$

其中 a_0,a_1 和 a_2 是常数。让我们考虑更一般的情况:假设我们有对于 $0\le\text{I}\le2$,$0\le\text{J}\le4$,$0\le\text{K}\le10$,$0\le\text{L}\le2$ 的一字元素 Q[I,J,K,L]的四维数组,我们将把存储器分配成使

得

$$\mathrm{LOC}(\mathrm{Q}[\mathrm{I},\mathrm{J},\mathrm{K},\mathrm{L}]) = a_0 + a_1\mathrm{I} + a_2\mathrm{J} + a_3\mathrm{K} + a_4\mathrm{L} \tag{3}$$

这意味着，I，J，K 或 L 的一个改变会导致对于 Q[I,J,K,L]的地址的容易计算的改变。分配存储器的最自然(因而也是最普遍使用的)方法是按照它们的下标的字典序来安排数组元素(习题 1.2.1-15d))，有时称做"行居先顺序"：

Q[0,0,0,0]，Q[0,0,0,1]，Q[0,0,0,2]，Q[0,0,1,0]，Q[0,0,1,1]，…，
Q[0,0,10,2]，Q[0,1,0,0]，…，Q[0,4,10,2]，Q[1,0,0,0]，…，
Q[2,4,10,2]

容易看出，这个顺序满足(3)的要求，因此有

$$\mathrm{LOC}(\mathrm{Q}[\mathrm{I},\mathrm{J},\mathrm{K},\mathrm{L}]) = \mathrm{LOC}(\mathrm{Q}[0,0,0,0]) + 165\mathrm{I} + 33\mathrm{J} + 3\mathrm{K} + \mathrm{L} \tag{4}$$

一般说来，给定对于

$$0 \leqslant \mathrm{I}_1 \leqslant d_1, 0 \leqslant \mathrm{I}_2 \leqslant d_2, \cdots, 0 \leqslant \mathrm{I}_k \leqslant d_k$$

的 c 个字的元素 $\mathrm{A}[\mathrm{I}_1,\mathrm{I}_2,\cdots,\mathrm{I}_k]$的一个 k 维数组，我们可以把它存入内存如下：

$$\begin{aligned}&\mathrm{LOC}(\mathrm{A}[\mathrm{I}_1,\mathrm{I}_2,\cdots,\mathrm{I}_k]) = \\ &\quad \mathrm{LOC}(\mathrm{A}[0,0,\cdots,0]) + c(d_2+1)\cdots(d_k+1)\mathrm{I}_1 + \cdots + c(d_k+1)\mathrm{I}_{k-1} + c\mathrm{I}_k = \\ &\quad \mathrm{LOC}(\mathrm{A}[0,0,\cdots,0]) + \sum_{1\leqslant r\leqslant k} a_r \mathrm{I}_r\end{aligned} \tag{5}$$

其中

$$a_r = c\prod_{r<s\leqslant k}(d_s+1) \tag{6}$$

为了看出为什么这个公式是有效的，观察一下，如果 $\mathrm{I}_1,\cdots,\mathrm{I}_r$ 是常数，而且 $\mathrm{J}_{r+1},\cdots,\mathrm{J}_k$ 遍历 $0\leqslant \mathrm{J}_{r+1}\leqslant d_{r+1},\cdots,0\leqslant \mathrm{J}_k\leqslant d_k$ 的所有值，则 a_r 是为存储子数组 $\mathrm{A}[\mathrm{I}_1,\cdots,\mathrm{I}_r,\mathrm{J}_{r+1},\cdots,\mathrm{J}_k]$所需要内存的数量；因此由字典序的特性，$\mathrm{A}[\mathrm{I}_1,\cdots,\mathrm{I}_k]$的地址在 I_r 改变 1 时，应精确地按这个数量来改变。

在一个混合进制的系统中公式(5)和(6)对应于数 $\mathrm{I}_1\mathrm{I}_2\cdots\mathrm{I}_k$ 的值。例如，如果对于 $0\leqslant \mathrm{W}<4, 0\leqslant \mathrm{D}<7, 0\leqslant \mathrm{H}<24, 0\leqslant \mathrm{M}<60$ 以及 $0\leqslant \mathrm{S}<60$，有数组 TIME[W,D,H,M,S]，则 TIME[W,D,H,M,S]的地址应是 TIME[0,0,0,0,0]的地址加上"W 周 + D 天 + H 小时 + M 分 + S 秒"转换成秒的数量。当然，利用有2 419 200个元素的数组是很费心的。

当数组有一个完全矩形的结构时，存储数组的通常方法一般就合适了，使得所有元素 $\mathrm{A}[\mathrm{I}_1,\mathrm{I}_2,\cdots,\mathrm{I}_k]$对于独立范围 $l_1\leqslant \mathrm{I}_1\leqslant u_1, l_2\leqslant \mathrm{I}_2\leqslant u_2,\cdots,l_k\leqslant \mathrm{I}_k\leqslant u_k$ 中的下标出现。习题 2 说明如何对(5)和(6)进行修改以适合于当下限$(l_1,l_2,\cdots,l_k)$不是$(0,0,\cdots,0)$时的情况。

但是还有很多其中数组不是一个完全矩形的情况。最普遍的是三角矩阵，其中我们要存入的仅仅是对于比如说 $0\leqslant k\leqslant j\leqslant n$ 的元素 $\mathrm{A}[j,k]$：

$$\begin{pmatrix} \mathrm{A}[0,0] & & & \\ \mathrm{A}[1,0] & \mathrm{A}[1,1] & & \\ \vdots & \vdots & \ddots & \\ \mathrm{A}[n,0] & \mathrm{A}[n,1] & \cdots & \mathrm{A}[n,n] \end{pmatrix} \tag{7}$$

我们可以知道,所有其它的元素都为零,或者 $A[j,k]=A[k,j]$,使得只需要存储一半的值就够了。如果要在$\frac{1}{2}(n+1)(n+2)$个连续的存储单元中保存下三角矩阵(7),我们就被迫放弃像在等式(2)中那样的线性分配的可能性,但可以要求代之以形如

$$\mathrm{LOC}(\mathrm{A}[\mathrm{J},\mathrm{K}]) = a_0 + f_1(\mathrm{J}) + f_2(\mathrm{K}) \tag{8}$$

的分配安排,其中 f_1 和 f_2 是单变量函数。(如果需要,常数 a_0 可以被吸收到 f_1 或 f_2 当中。)当地址有(8)的形式时,如果我们保持有两个(稍微短的)f_1 和 f_2 的值的辅助表,就可以迅速地访问随机元素 A[J,K];因此这些函数仅须计算一次。

结果是,对于数组(7)的下标的字典序满足条件(8),而且对于一个字的项,我们事实上有简单的公式

$$\mathrm{LOC}(\mathrm{A}[\mathrm{J},\mathrm{K}]) = \mathrm{LOC}(\mathrm{A}[0,0]) + \frac{\mathrm{J}(\mathrm{J}+1)}{2} + \mathrm{K} \tag{9}$$

但是如果足够幸运,有两个同样大小的三角矩阵,我们实际上有好得多的方法来保存它们。假设对于 $0\leq k\leq j\leq n$,要保存 $A[j,k]$和 $B[j,k]$,那么我们就可以把它们两者都放在单个矩阵 $C[j,k]$中,$0\leq j\leq n$,$0\leq k\leq n+1$,使用以下约定

$$A[j,k] = C[j,k], \qquad B[j,k] = C[k,j+1] \tag{10}$$

于是

$$\begin{pmatrix} C[0,0] & C[0,1] & C[0,2] & \cdots & C[0,n+1] \\ C[1,0] & C[1,1] & C[1,2] & \cdots & C[1,n+1] \\ \vdots & & & & \vdots \\ C[n,0] & C[n,1] & C[n,2] & \cdots & C[n,n+1] \end{pmatrix} \equiv \begin{pmatrix} A[0,0] & B[0,0] & B[1,0] & \cdots & B[n,0] \\ A[1,0] & A[1,1] & B[1,1] & \cdots & B[n,1] \\ \vdots & & & & \vdots \\ A[n,0] & A[n,1] & A[n,2] & \cdots & B[n,n] \end{pmatrix}$$

这就把两个三角矩阵一起紧密地组装在$(n+1)(n+2)$个单元的空间内,于是我们就有像(2)中那样的线性编址。

把三角矩阵推广到更高的维数上称做*四面体数组*。这一课题是习题 6 到习题 8 的主题。

使用顺序地保存的数组的典型程序设计技术的例子,请参见习题 1.3.2-10 和对该习题给出的两个答案。那些程序中涉及的有效遍历行和列与使用顺序栈的基本技术,是特别有用的。

链接分配　链接存储分配的技术自然也适用于更高维的信息数组。一般来说,我们的节点可以包含 k 个链接字段,每个字段表示该节点所属的每个表。使用链接存储一般是对其中数组并不严格具有矩形特征的情况进行的。

作为一个例子,我们可以有每个节点表示一个人的一个表,并有四个链接字段:SEX(性别),AGE(年龄),EYES(眼睛),以及 HAIR(头发)。在 EYES 字段中,我们把具有相同眼睛颜色的所有节点链接在一起,等等。(见图 13。)容易想像把一个新的人插入到这个表的有效算法;然而删除就将慢得多,除非我们使用双重链接。我们也可以想像进行如"找出年龄从 21 到 23 的所有蓝眼睛金黄发女人"这样的事情的效率程度不同的一些算法;见习题 9 和 10。一个表的每个节点同时驻留在若干类型的其它表中的问题相当经常地出现;其实,在上一小节中描述的电梯系统的模拟就有同时在 QUEUE 和

WAIT 表两者当中的节点。

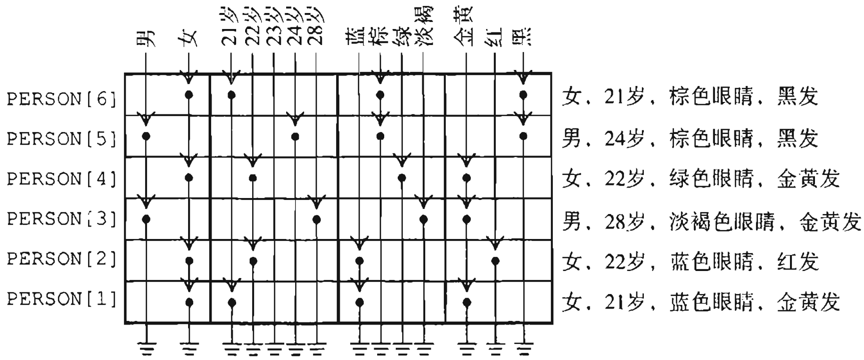


图 13 在四种不同情况下的每个节点

作为对正交表使用链接分配的一个详尽例子,我们将考虑稀疏矩阵的情况(即大多数元素为零的高阶矩阵)。目标是对这些矩阵进行操作如同整个矩阵都存在一样,但是由于不需要表示零元素而节省大量的时间和空间。做这件事的一个方法,旨在对矩阵元素进行随机访问,将是使用第 6 章的存储和查找方法,来从关键字“$[j,k]$”找出 A$[j,k]$;然而,还有另一个处理稀疏矩阵的方法,由于它更适当地反映矩阵结构,通常是更可取的。因此是我们在这里将要讨论的方法。

我们将要讨论的表示法由对每个行和每个列循环链接的表组成,矩阵的每个节点包含三个字和五个字段:

ROW	UP
COL	LEFT
VAL	

(11)

这里,ROW 和 COL 是节点的行和列的下标;VAL 是存储在矩阵该部分的值;LEFT 和 UP 分别是对这行左边下一个非零项的链接,或者对列的往上的下一个非零项的链接。对于每行和每列,有特殊的表头节点 BASEROW$[i]$和 BASECOL$[j]$。这些节点由

$$\text{COL}(\text{LOC}(\text{BASEROW}[i])) < 0 \text{ 及 } \text{ROW}(\text{LOC}(\text{BASECOL}[j])) < 0$$

所标识。像通常在一个循环表中一样,BASEROW$[i]$中的 LEFT 链接是在该行中最右值的地址。而在 BASECOL$[j]$中 UP 指向在该列中最底部的值。例如,矩阵

$$\begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix} \tag{12}$$

将如图 14 所示那样表示。

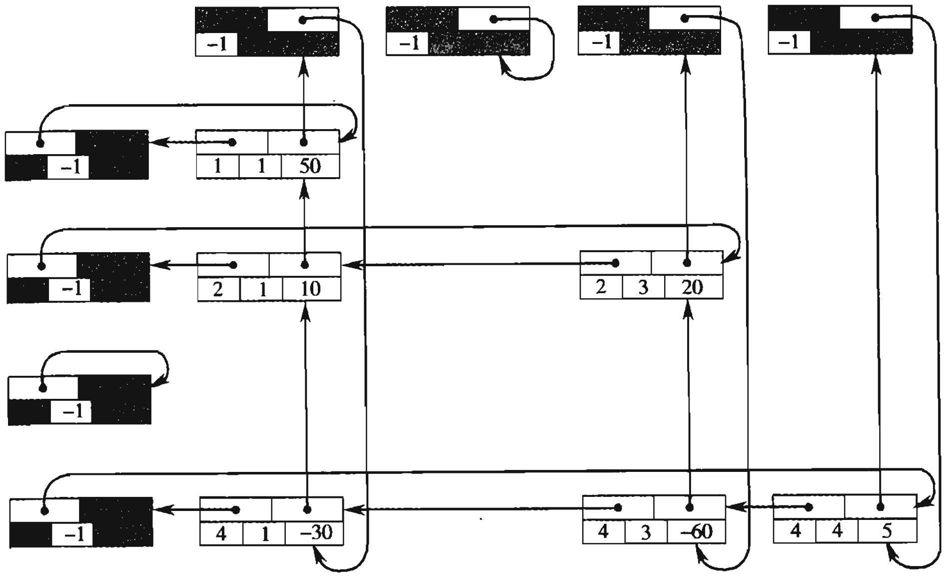


图 14　矩阵(12)的表示，且节点有格式为 LEFT | UP / ROW | COL | VAL 。表头出现在左边和顶部

使用存储器的顺序分配，一个 200×200 的矩阵将占用 40000 个字，这比许多计算机一向拥有的内存还多；但是一个适度稀疏的 200×200 的矩阵甚至在 MIX 的 4000 字的内存中也可以像上面那样表示。(参见习题 11。)花费在访问一个随机元素 A[j, k]的时间数量也是十分合理的，如果在每行或列中只有少量元素的话；由于大多数的矩阵算法都是通过顺序走过一个矩阵进行的，而不是随机地来访问元素，这个链接表示通常要比顺序链接的表示工作得更快。

作为处理这种形式下的稀疏矩阵的一个非平凡算法的例子，我们将考虑*主元步骤*的操作，它是解决线性方程、矩阵求逆以及通过单纯形法解决线性规划问题的算法的重要部分。一个主元步骤是下列矩阵变换：

$$
\begin{array}{l}
\\ \\ \text{主元行} \\ \text{任何其它行} \\ \\
\end{array}
\overset{\textit{在主元步骤之前}}{\overset{\qquad\quad\text{主元列}\quad\text{任何其它列}}{\begin{pmatrix}
 & \vdots & & \vdots & \\
\cdots & a & \cdots & b & \cdots \\
 & \vdots & & \vdots & \\
\cdots & c & \cdots & d & \cdots \\
 & \vdots & & \vdots &
\end{pmatrix}}},
\overset{\textit{在主元步骤之后}}{\overset{\text{主元列}\qquad\text{任何其它列}}{\begin{pmatrix}
 & \vdots & & \vdots & \\
\cdots & 1/a & \cdots & b/a & \cdots \\
 & \vdots & & \vdots & \\
\cdots & -c/a & \cdots & d-bc/a & \cdots \\
 & \vdots & & \vdots &
\end{pmatrix}}}
\qquad (13)
$$

假定主元 a 非零。例如，以 2 行 1 列的 10 作为主元，应用主元步骤于矩阵(12)，导致

$$\begin{pmatrix} -5 & 0 & -100 & 0 \\ 0.1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 5 \end{pmatrix} \tag{14}$$

我们的目标是设计一个算法,对如图 14 所示的稀疏矩阵实施这一主元操作。显然,转换(13)仅影响一个矩阵的主元列中有非零元素的那些行,以及在主元行中有非零元素的那些列。

主元算法在许多方面是我们已经讨论的链接技术的一个直截了当的应用;特别是,它同多项式加法的算法 2.2.4A 有很强的类似性。然而,有两件事情使这个问题有点奇巧:如果在(13)中,我们有 $b \neq 0$ 和 $c \neq 0$,但 $d = 0$,则稀疏矩阵对 d 就没有项,因而我们必须插入一个新项;而如果 $b \neq 0, c \neq 0, d \neq 0$,但 $d - bc/a = 0$,则我们必须删去原来在那里的项。这些插入和删除的操作在二维数组的情况下要比在一维数组的情况下更有趣;要实现这些操作我们必须知道受影响的是什么链接。我们的算法是由底向上逐行地对矩阵进行的。插入和删除的有效能力涉及引进一组指针变量 PTR[j],对所考虑的每列使用一个指针。这些指针向上遍历这些列,给了我们在两个维中更新适当链接的能力。

算法 S(*在一个稀疏矩阵中的主元步骤*) 给定如图 14 所示的矩阵,我们实施主元操作(13)。假定 PIVOT 是指向主元的一个链接变量。本算法利用链接变量 PTR[j]的一个辅助表,对于矩阵的每列有一个指针。变量 ALPHA 和每个节点的 VAL 字段假定都是浮点数或有理量,而本算法中其它的每个量都有整数值。

S1.[初始化] 置 ALPHA←1.0/VAL(PIVOT),VAL(PIVOT)←1.0 和

I0 ← ROW(PIVOT),P0 ← LOC(BASEROW[I0]);

J0 ← COL(PIVOT),Q0 ← LOC(BASECOL[J0])

S2.[处理主元行] 置 P0←LEFT(P0),J←COL(P0)。如果 J < 0,则转步骤 S3(主元行已被遍历)。否则置 PTR[J]←LOC(BASECOL[J])并且 VAL(P0)←ALPHA × VAL(P0),并重复步骤 S2。

S3.[找新行] 置 Q0←UP(Q0)。(本算法的剩余部分由底向上地逐个处理每个行,在这些行的主元列处有一个元素。)置 I←ROW(Q0)。若 I < 0,算法终止。若 I = I0,重复步骤 S3(我们已经完成主元行)。否则置 P←LOC(BASEROW[I]),P1←LEFT(P)。(指针 P 和 P1 现在将从右到左地穿过各行,如同 P0 同步地穿过行 I0 一样;算法 2.2.4A 类似。这时我们有 P0 = LOC(BASEROW[I0]。)

S4.[找新列] 置 P0←LEFT(P0),J←COL(P0). 如果 J < 0,则置 VAL(Q0)← − ALPHA × VAL(Q0)并返回步骤 S3。如果 J = J0,则重复步骤 S4。(因此在处理完所有其它列的元素之后,我们处理在行 I 中的主元列元素;原因是在步骤 S7 中需要 VAL(Q0)。)

S5.[找 I,J 元素] 如果 COL(P1) > J,置 P←P1,P1←LEFT(P),并重复步骤 S5。如果 COL(P1) = J,转到步骤 S7。否则转到步骤 S6(我们需要在 I 行 J 列处插入

一个新元素)。

S6. [插入 I,J 元素] 如果 ROW(UP(PTR[J])) > I,置 PTR[J]←UP(PTR[J]),并且重复步骤 S6。(否则我们将有 ROW(UP(PTR[J])) < I;要把新元素插入到垂直的维数中刚好在 NODE(PTR[J])的上边,在水平维数中刚好在 NODE(P)的左边。否则置 X⇐AVAIL,VAL(X)←0,ROW(X)←I,COL(X)←J,LEFT(X)←P1,UP(X)←UP(PTR[J]),LEFT(P)←X,UP(PTR[J])←X,P1←X。

S7. [主元] 置 VAL(P1)←VAL(P1) - VAL(Q0) × VAL(P0)。如果现在 VAL(P1) = 0,转到步骤 S8。(注:当使用浮点数算术时,测试"VAL(P1) = 0"应当由"|VAL(P1)| < EPSILON"或者更好地由条件"在减法中 VAL(P1)的大多数有效数字都失去"所代替。)否则,置 PTR[J]←P1,P←P1,P1←LEFT(P),并返回 S4。

S8. [删去 I,J 元素] 如果 UP(PTR[J]) ≠ P1 (或者,实际上是同一回事,如果 ROW(UP(PTR[J])) > I),置 PTR[J]←UP(PTR[J])并且重复步骤 S8;否则,置 UP(PTR[J])←UP(P1),LEFT(P)←LEFT(P1),AVAIL⇐P1,P1←LEFT(P)。转回 S4。▌

作为一个非常有益的习题,把本算法的程序设计留给作者完成(见习题 15)。需要指出的是,由于每一个节点 BASEROW[i],BASECOL[j]的大多数字段都是不相关的,因此对每个节点只需分配一个字的内存。(请见图 14 中的带阴影的区域,并参见 2.2.5 小节的程序。)其次,为了进一步节省存储空间,值 - PTR[j]可以存储为 ROW(LOC(BASECOL[j]))。算法 S 的运行时间非常粗略地同受主元操作影响的矩阵元素个数成比例。

这种通过正交循环表对稀疏矩阵的表示是有益的,但数值分析已经发展了更好的方法。请参见 Fred. G. Gustavson, *ACM Trans. on Math. Software* **4** (1978),250 ~ 269;也请参见第 7 章中的图形和网络算法。

习　题

1. [*17*] 如果 A 是(1)的矩阵,而且数组的每一个节点是两个字长,并且假定节点以其下标的字典序储存,给出 LOC(A[J,K])的公式来。

▶ 2. [*21*] 公式(5)和(6)已从对于 $1\leqslant r\leqslant k$,假定 $0\leqslant \mathrm{I}_r\leqslant d_r$ 而被导出;给出可适用于情况 $l_r\leqslant \mathrm{I}_r\leqslant u_r$ 的一般公式,其中 l_r 和 u_r 是维数的任何下限和上限。

3. [*21*] 正文考虑对于 $0\leqslant k\leqslant j\leqslant n$ 的下三角矩阵 A[j,k]。对这样的矩阵的讨论如何能修改成下标由 1 开始而不是 0 开始的情况,使得 $1\leqslant k\leqslant j\leqslant n$?

4. [*22*] 说明如果对于 $0\leqslant k\leqslant j\leqslant n$ 我们以下标的字典序存储上三角矩阵 A[j,k],这个分配满足等式(8)的条件。在这个意义下找出 LOC(A[J,K])的公式。

5. [*20*] 说明利用习题 2.2.2-3 中的间接寻址特征,有可能在一条 MIX 指令中把 A[J,K]的值放进寄存器 A 中,甚至当 A 如同在(9)中那样是一个三角矩阵时。(假定 J 和 K 的值都在变址寄存器中。)

▶ 6. [*M24*] 考虑"四面体数组"A[i,j,k],B[i,j,k],其中在 A 里有 $0\leqslant k\leqslant j\leqslant i\leqslant n$,在 B 里有

$0 \leqslant i \leqslant j \leqslant k \leqslant n$。假定把这两个数组都按下标的字典序存放在连续的内存单元中；证明对于某些函数 f_1, f_2, f_3，$\mathrm{LOC}(\mathrm{A}[\mathrm{I},\mathrm{J},\mathrm{K}]) = a_0 + f_1(\mathrm{I}) + f_2(\mathrm{J}) + f_3(\mathrm{K})$。能否以类似方式表达 LOC(B[I,J,K])？

7. [*M23*] 找出对于 k 维四面体数组 $\mathrm{A}[i_1, i_2, \cdots, i_k]$ 分配存储器的一个一般公式，其中 $0 \leqslant i_k \leqslant \cdots \leqslant i_2 \leqslant i_1 \leqslant n$。

8. [*33*] (P. Wegner) 假定有六个四面体数组 A[I,J,K]，B[I,J,K]，C[I,J,K]，D[I,J,K]，E[I,J,K]和 F[I,J,K]要存到内存中，其中 $0 \leqslant \mathrm{K} \leqslant \mathrm{J} \leqslant \mathrm{I} \leqslant n$。有没有类似于二维情况(10)那样实现这一点的干净利索的方法？

9. [*22*] 假定已经建立了类似于图 13 所指出形式的一个表格，但要比那个还大得多，使得像在那里所示所有的链接都按相同方向进行（即，对于所有节点和链接，LINK(X) < X)。试设计一个算法，找出年龄从 21 岁到 23 岁的所有蓝眼睛金黄发的女人的地址，它通过这样的方式遍历各种链接字段，即在算法执行完成时，至多对每个表 FEMALE，A21，A22，A23，BLOND 和 BLUE 扫描一次。

10. [*26*] 你能否以更好的方式来组织人员的表格，使得对上道题所描述的查找将更为有效？(对本题的答案不仅仅是“是”或者“否”。)

11. [*11*] 假设有一个每行至多四个非零元素的 200×200 的矩阵。如果除了表头之外每个节点使用三个字，表头使用一个字。为了如同在图 14 中那样表示这个矩阵，要求多少存储单元？

▶ 12. [*20*] 借助于在(13)中所用记号 a, b, c, d，在步骤 S7 开始处的 VAL(Q0)，VAL(P0)及 VAL(P1)是什么？

▶ 13. [*22*] 为什么在图 14 中使用循环表而不使用直接的线性表？能否重写算法 S 使得它不利用循环链接？

14. [*22*] 算法 S 实际上节省了在一个稀疏矩阵中主元步骤的时间，因为它避免考虑主元行有一个零元素的那些列。说明通过一个辅助表 LINK(j)，$1 \leqslant j \leqslant n$ 的帮助，在顺序地存储的一个很大的稀疏矩阵中，可以节省运行时间。

▶ 15. [*29*] 编写算法 S 的一个 MIXAL 程序。假定 VAL 字段是一个浮点数，而且对这个字段的运算，可以使用浮点算术运算符 FADD，FSUB，FMUL 和 FDIV。为简便起见，假定当被加或被减的操作数消去大多数有效位时，FADD 和 FSUB 返回零的答案，使得在步骤 S7 中可以安全地使用“VAL(P1) = 0”的测试。浮点运算仅使用 rA，不使用 rX。

16. [*25*] 试设计一个算法复制一个稀疏矩阵。(换句话说，开始时仅仅给出矩阵的一种表示，本算法在内存中生成矩阵的另一种不同的表示，且它们都有图 14 的形式。)

17. [*26*] 试设计把两个稀疏矩阵相乘的一个算法；给定矩阵 A 和 B，形成一个新矩阵 C，其中 $\mathrm{C}[i,j] = \sum_k \mathrm{A}[i,k]\mathrm{B}[k,j]$。两个输入矩阵和输出矩阵应如图 14 中那样表示。

18. [*22*] 假定对于 $1 \leqslant i, j \leqslant n$，矩阵的元素是 $\mathrm{A}[i,j]$，下列算法以矩阵的逆来代替该矩阵。

i)对于 $k = 1, 2, \cdots, n$，执行下列操作：在未被用作主元列的所有列中，查找行 k，以找出有最大绝对值的元素；置 C[k]等于找到这个元素的列，并且以这个元素作为主元执行一个主元步骤。(如果所有这样的元素都为零，则这矩阵为奇异的，因而没有逆矩阵。)

ii)转换行和列，使得行 k 变成行 C[k]，列 C[k]变成列 k。

本习题中的问题是使用上述算法，通过手算，来对矩阵

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

求逆。

19. [*31*] 修改习题 18 中的算法,使得它得到以图 14 中的形式表示的一个稀疏矩阵的逆。特别要注意使步骤 ii)的行和列的转换运算有效。

20. [*20*] 三对角矩阵有这样的项 a_{ij},即对于 $1 \le i, j \le n$,除当 $|i-j| \le 1$ 外,a_{ij} 皆为零。说明有一个形如

$$\mathrm{LOC}(\mathrm{A}[\mathrm{I},\mathrm{J}]) = a_0 + a_1\mathrm{I} + a_2\mathrm{J}, \qquad |\mathrm{I}-\mathrm{J}| \le 1$$

的分配函数,它在$(3n-2)$个连续的单元中表示一个三对角矩阵的所有有关的元素。

21. [*20*] 提出一个对 $n \times n$ 矩阵的存储分配函数,其中 n 是变量。不管 n 的值是什么,对于 $1 \le i, j \le n$,元素 A[I,J]应该占据 n^2 个连续的单元。

22. [*M25*] (P. Chowla,1961) 找出一个多项式 $p(i_1,\cdots,i_k)$,当下标$(i_1,\cdots,i_k)$取遍所有 k 维非负整向量时,它取每个非负整数值恰一次,而且还具有附加的性质,即 $i_1+\cdots+i_k < j_1+\cdots+j_k$,意味着 $p(i_1,\cdots,i_k) < p(j_1,\cdots,j_k)$。

23. [*23*] 一个可扩充矩阵开始时为 1×1,然后通过加上新的行或列,它从 $m \times n$ 的大小变成$(m+1) \times n$ 或 $m \times (n+1)$的大小。证明这样一个矩阵可给予一个简单的分配函数,在此函数之下,对于 $0 \le \mathrm{I} \le m$ 和 $0 \le \mathrm{J} < n$,元素 A[I,J]占据 mn 个连续的单元。当矩阵增长时,没有元素改变地址。

▶ 24. [*25*] (稀疏数组技巧) 假设你要使用一个大的数组以进行随机存取,尽管你实际上将不去访问它的很多的元素。你要求在你头一次访问 A[k]时它为零,但你又不要花费时间来把每个位置都设置为零。试说明如何不必对实际的初始内存内容作任何假定,而只对每个数组访问作少量固定的附加操作,就可以对给定的 k 可靠地读和写任何要求的单元 A[k]。

2.3 树

我们现在转到对于树——在计算机算法中出现的最重要的非线性结构——的研究。一般地说,树结构意味着节点间的一个"分支"关系,很像在天然的树中见到的那样。

让我们形式地把树定义为一个或多个节点的有限集合 T,使得

a)有一个特别指定的节点,叫做树的根 root(T);以及

b)剩余的节点(排除根)被分划成 $m\geqslant 0$ 个不相交的集合 $T_1,\cdots,T_m$,而且这些集合的每一个也都是树。树 $T_1,\cdots,T_m$ 称做这个根的*子树*。

刚才给出的定义是递归的,我们用树定义了树。当然,这里不涉及循环的问题,因为具有一个节点的树必须只由一个根组成,而有 $n>1$ 个节点的树是借助于具有少于 n 个节点的树来定义的。因此一棵具有两个、三个或最终任何数目节点的树是由给定的定义确定的。也有定义树的非递归的方法(例如,见习题 10,12 和 14,以及 2.3.4 小节)。但是递归定义似乎最合适,因为递归是树结构天生的特征。树的递归特征也在自然界中出现,因为幼树的树芽逐渐地长出有它们自己的树芽的子树,等等。习题 3 说明如何基于如同上面给出的递归定义,利用对一个树中节点个数的归纳法,来给出关于树的重要事实的严格证明。

从我们的定义得出,树的每一个节点是被包含在整个树当中的某个子树的根。一个节点的子树的个数称做该节点的*度*。度数为零的节点称做*终(端)节点*,或者有时叫做*叶*。一个非终节点通常称做一个*分支节点*。相对于 T,一个节点的*级*递归地定义为:根 root(T)的级为 0,任何其它节点的级相对于包含该节点的 root(T)的子树的对应节点的级大 1。

这些概念图解于图 15 中,该图示出一个具有七个节点的树。根是 A,而它有两个子树 $\{B\}$ 和 $\{C,D,E,F,G\}$。树 $\{C,D,E,F,G\}$ 以 C 为根,相对于整棵树节点 C 在 1 级上,而它有三个子树 $\{D\}$,$\{E\}$ 和 $\{F,G\}$;因此 C 有度数 3。图 15 中的终端节点是 B,D,E 和 G;F 是惟一有度数 1 的节点,G 是惟一有级数 3 的节点。

如果子树 $T_1,\cdots,T_m$ 在定义的 b)中的相对顺序重要的话,我们就说该树是*有序树*;当在一个有序树中 $m\geqslant 2$ 时,把 T_2 称做根的"第二棵子树"等等,是有意义的。有序树也被一些作者称做"平面树",因为它与在一个平面中嵌入树的方式有关。如果不把仅仅是节点子树的相对次序不同的两棵树当做不同的树,就说这样的树是*有向的*,因为考虑的仅仅是节点相对的方向,而不是它们的次序。计算机表示的本质定义了任何树的一种隐含顺序,因此在大多数情况下,有序树最令我们感兴趣。所以我们将不言而喻地假定,*讨论的所有树都是有序的,除非另有明确的说明*。因此,图 15 和 16 的树一般地认为是不同的,尽管作为有向树它们是相同的。

*森林*是零个或多个不相交的树的集合(通常是一个有序的集合)。表达定义的 b)部分的另一种方式是说*树的除了根之外的所有节点形成森林*。

在抽象的森林和树之间的区别很小。如果删去树的根,我们就有一个森林;反过来,如果在任何森林上增加一个节点,并且把森林中的树当做这个新节点的子树,我们

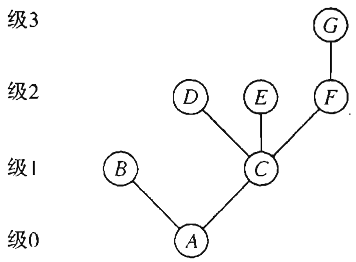


图 15 一棵树

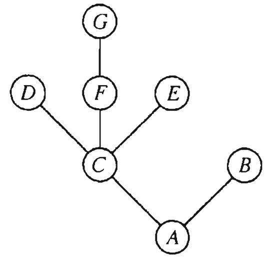


图 16 另一棵树

就得到一棵树。因此在关于数据结构的非正式讨论当中,树和森林通常几乎是可以互换的。

有许多方式来画树。除图 15 中的表示外,根据根放置地方的不同,图 17 示出三种主要的选择。关心如何以图形来画出树结构并非是无聊的笑话,因为在许多场合下我们要说一个节点在另外一个节点"上面"或者"高于"另一个节点,或者参照"最右"的元素等等。处理树结构的某些算法已经公认为"由顶向下"的算法,而不是反方向的"由底向上"。除非我们坚持画树的一致约定,不然这样的术语就会导致混乱。

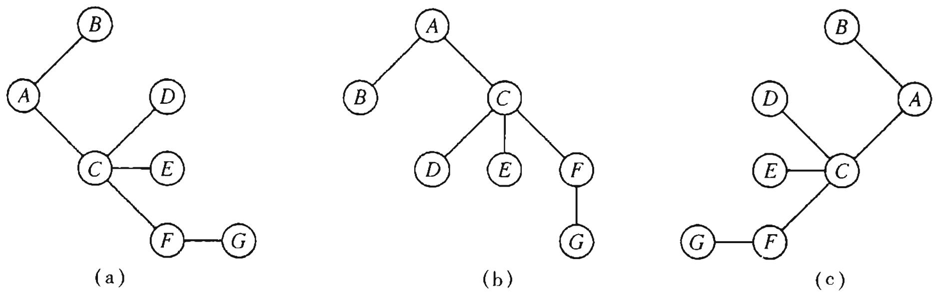


图 17 我们应如何画一棵树?

看起来图 15 的形式是可取的,因为它符合于树的自然生长情况;如果没有原因强迫采用其它三种形式中的任何一种,我们也会采用自然的由来已久的传统。就由于牢记这一点,作者在编写这套书时曾经始终坚持根在下部的约定。但在尝试两年之后,发现这是一个错误。对计算机著作的考察以及同计算机科学家进行关于广泛的各种各样算法的许多非正式的讨论表明,以根在顶部来画树占了被考察情况的百分之八十以上。使手画的图表向下生长而不是向上(就我们写东西的方式来看,这是容易理解的)具有压倒性的优势;甚至子树(subtree)一词,与"上方树"(supertree)相反,也倾向于隐含一个向下的关系。从这些考虑出发,我们得出结论:*图 15 是画颠倒了*。从今往后,我们将几乎总是如图 17(b)那样画树,把根画在顶部而把叶画在下部。对应于这个方向,我们也许应该把根节点称做树的顶点,而且谈到节点时说是处于浅层或是深层。

谈到树,需要使用正确的描述性术语,而不要使用"上面"和"下面"等有些含糊的引用。我们一般使用来自于家谱的术语。图 18 示出两个普通类型的家谱图。两种类型

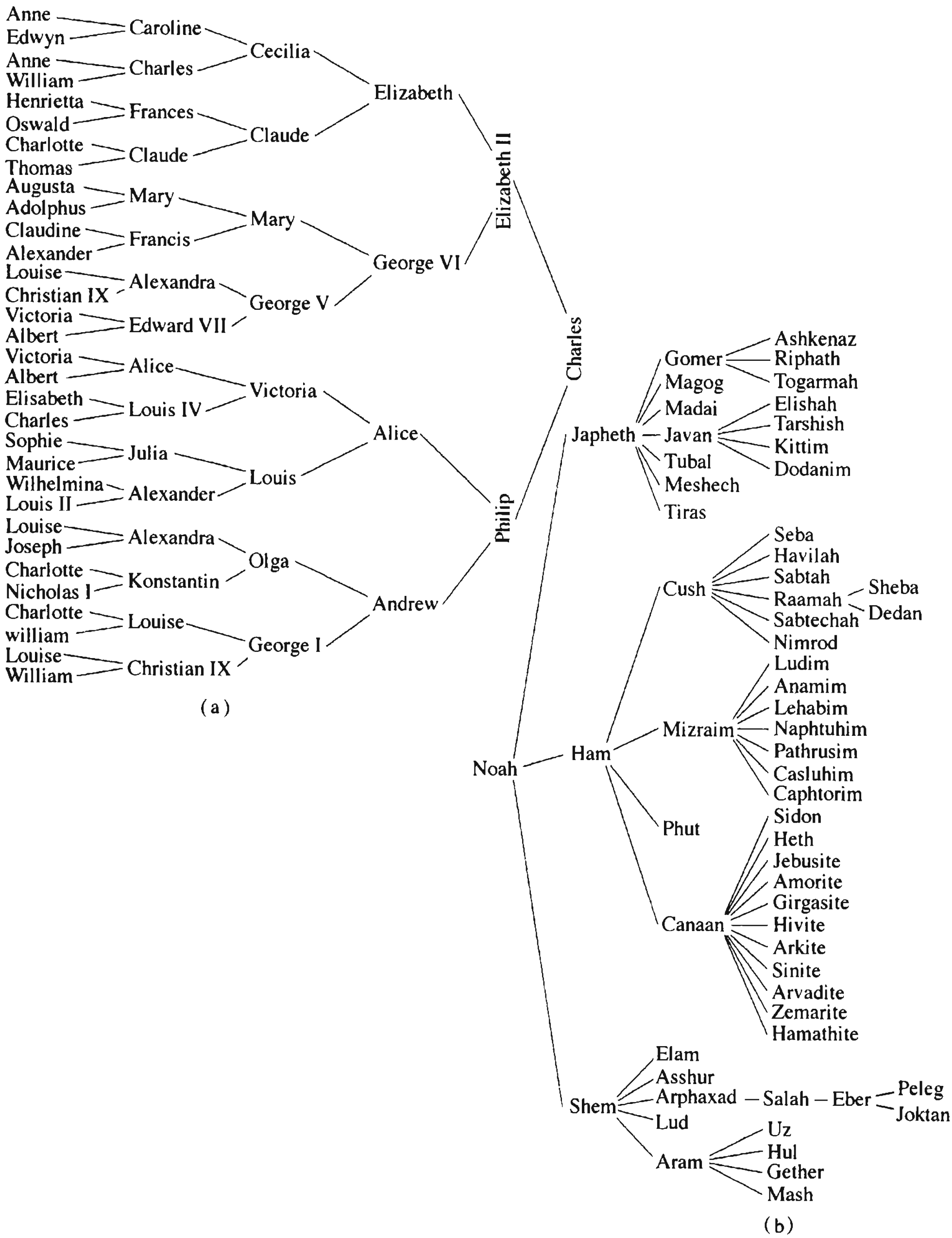


图 18　家谱图

(a)血统图;(b)世系图。

[参看:Burke 的*Peerage*(1959);*Almanach de Gotha*(1871);
Genealogisches Handbuch des Adels:Fürstliche Häuser **1**; Genesis 10:1～25]

十分不同，血统图示出一个给定的个人的祖宗，而世系图示出后裔。

如果出现有“交叉繁育”的情况，血统图实际上不是一棵树，因为一棵树的不同分支（像我们已经对它定义的那样）绝不能相交在一起。为了补偿这个缺陷，图 18(a)在第六代中两次提到 Victoria 王后和 Albert 王子；Christian IX 国王和 Louise 王后实际上既出现在第五代也出现在第六代中。血统图可以当做一棵真正的树，如果它的每个节点不仅仅表示作为个体的一个人，而是表示“以某某人母亲或父亲的角色出现的人”的话。

对于树结构的标准术语取自于第二种类型的家谱图，即世系图：人们常说每个根是其子树的根的双亲，而称诸子树为同胞；它们是其双亲的子女，整个树的根没有双亲。例如，在图 19 中，C 有三个子女：D, E 和 F；E 是 G 的双亲，B 和 C 是同胞。对此术语进行扩充——例如，A 是 G 的曾祖；B 是 F 的姑姑或叔伯；H 和 F 是头一代堂表兄弟姊妹——这显然是可能的。某些作者使用男性的称谓“父亲、儿子、兄弟”以代替“双亲、子女、同胞”；另外一些作者使用“母亲、女儿、姐妹”。在任何情况下，一个节点顶多有一个双亲或先人。我们使用祖先和后裔的词来表示可以跨越树的很多代的关系，在图 19 中 C 的后裔是 D, E, F 和 G；而 G 的祖先是 E, C 和 A。[*]

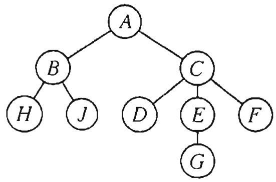
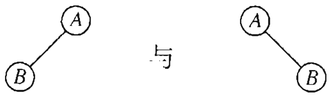


图 19　传统的树的图

图 18(a)中的血统图是二叉树的一个例子，它是树结构的另一种重要类型。读者毫无疑问已经见过与网球淘汰赛和其它体育事件相关联的二叉树。在一棵二叉树中，每个节点至多有两个子树；而当只有一棵子树时，我们区分左子树和右子树。更正式地说，让我们将二叉树定义为*节点的有限集合，它或者为空，或者由根和这个根的两个称做左和右子树的不相交二叉树的元素所组成*。

应该仔细地研究二叉树的这个递归定义：注意二叉树不是树的特殊情况；它全然是另外一个概念。（尽管我们将看到两个概念之间的许多联系。）例如，二叉树



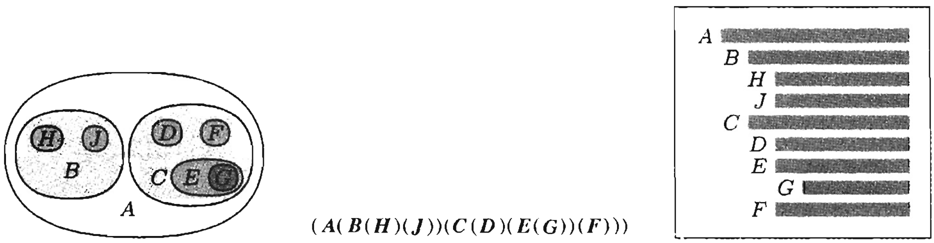
(1)

是不同的——在一种情况下根的右子树为空，而在另一种情况下，根有一个非空的右子树——尽管作为树，这些图将表示同样的结构。二叉树可以为空，树却不可以。因此我们总是小心地使用“二叉”来区分二叉树和普通的树。有些作者以稍微不同的方式来定义二叉树（见习题 20）。

可以以很多种其它方式用图形来表示树结构，它们使得树结构与实际的树不具类比性。图 20 示出反映图 19 的结构的三种框图：图 20(a)实际上把图 19 表示为一个有向树；这个框图是嵌套集合的一般思想的一个特殊情况，即集合的一个汇集，其中任何一对集合或者不相交或者一个包含另外一个。（见习题 10。）该图的(b)部分在一行里

* 鉴于中文中“双亲”、“父母”、“子女”等不便表达单数概念，且为了行文简捷，后续译文将多用“父亲”、“儿子”等，分别与原文的“parent”、“child”对应。——本书责任编辑

示出嵌套的集合,很像是(a)部分以一个平面表示它们那样;在(b)部分中还指出了树的顺序。(b)部分也可以当做是涉及嵌套的括弧的一个代数公式的表达。使用缩进方式,(c)部分仍然示出表示树结构的另一种普通方法。不同表示方法的种数本身是在日常生活中以及在计算机程序设计中树结构的重要性的丰富证据。任何层次分类方案都导致一个树结构。



(a) (b) (c)

图 20 表示树结构的进一步的方法

(a)嵌套集合;(b)嵌套的括弧;(c)缩进

一个代数公式定义一个隐含的树结构,它通常借助于替代使用括弧的手段,或者除了使用括弧之外的其它手段。例如,图 21 示出对应于算术表达式

$$a - b(c/d + e/f) \tag{2}$$

的一棵树。按照乘法和除法有比加法和减法高的优先权,标准的数学约定允许我们使用像(2)这样的一个简化的形式以代替完全括弧化的形式"$a-(b\times((c/d)+(e/f)))$"。在应用中公式和树的这种联系是非常重要的。

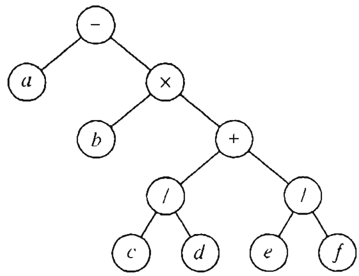
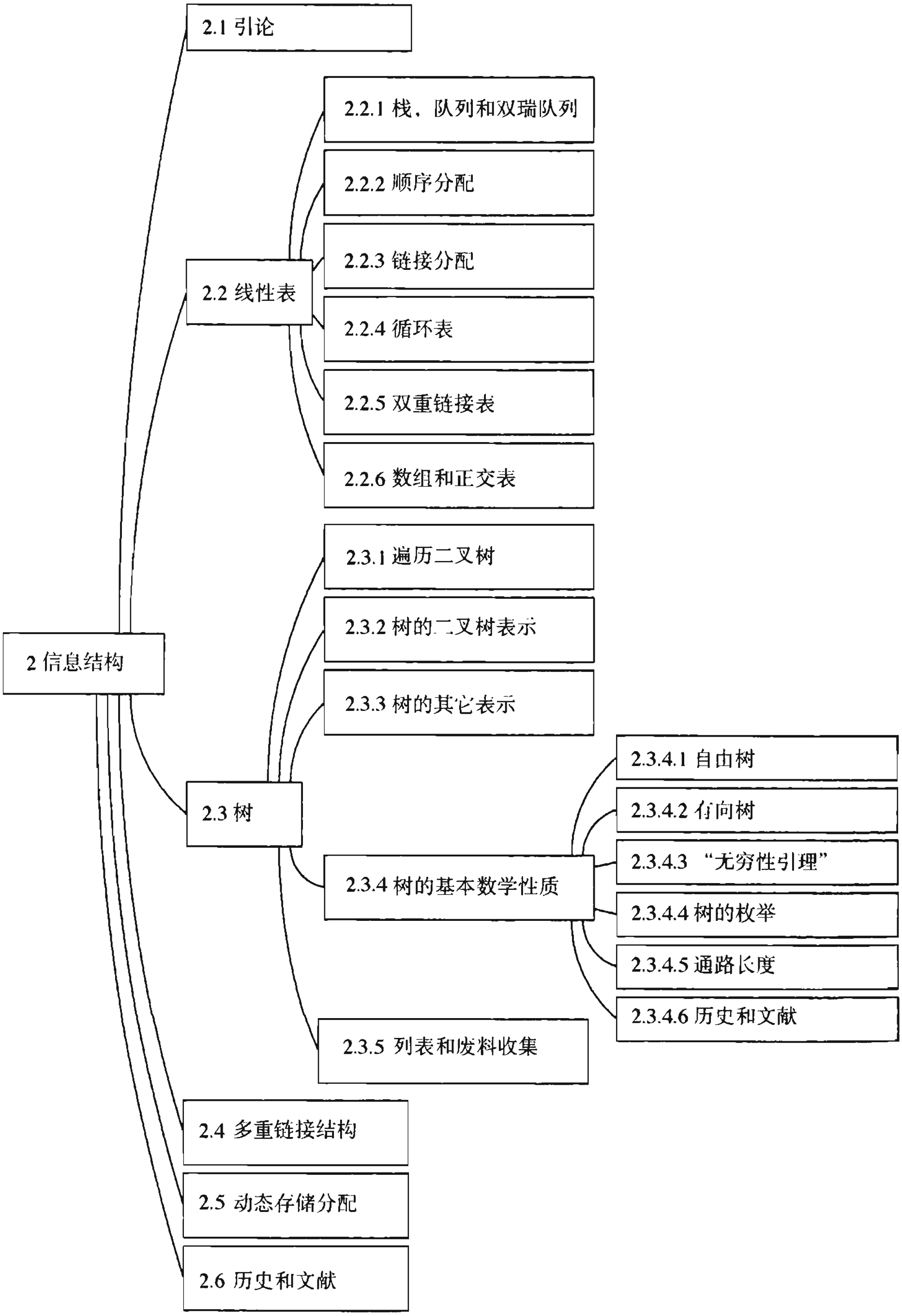


图 21 公式(2)的树表示

注意,图 20(c)中的缩进的表看起来非常像一本书的目录。其实,本书本身就有一个树结构;图 22 显示的是本书第 2 章的树结构。这里我们注意一个重要的思想:*在本书中用来对小节编号的方法是确定树结构的另一种方法*。根据与图书馆使用的同名的类似分类方案的类比,这样一个方法通常叫做树的"杜威(Dewey)十进记号"。对于图 19 的树的杜威十进记号是

1 *A*; 1.1 *B*; 1.1.1 *H*; 1.1.2 *J*; 1.2 *C*;
1.2.1 *D*; 1.2.2 *E*; 1.2.2.1 *G*; 1.2.3 *F*



2 信息结构
2.1 引论
2.2 线性表
2.2.1 栈，队列和双瑞队列
2.2.2 顺序分配
2.2.3 链接分配
2.2.4 循环表
2.2.5 双重链接表
2.2.6 数组和正交表
2.3 树
2.3.1 遍历二叉树
2.3.2 树的二叉树表示
2.3.3 树的其它表示
2.3.4 树的基本数学性质
2.3.4.1 自由树
2.3.4.2 有向树
2.3.4.3 “无穷性引理”
2.3.4.4 树的枚举
2.3.4.5 通路长度
2.3.4.6 历史和文献
2.3.5 列表和废料收集
2.4 多重链接结构
2.5 动态存储分配
2.6 历史和文献

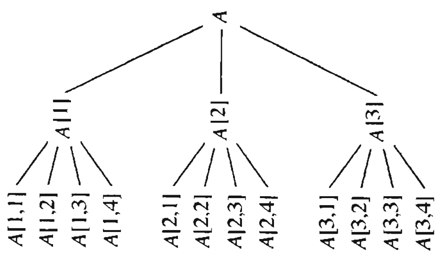
图 22　第 2 章的结构

杜威十进记号适用于任何森林:森林中的第 k 级树的根给予号码 k;而且如果 α 是度数为 m 的任何节点的号码,则它的儿子被编号为 $\alpha.1,\alpha.2,\cdots,\alpha.m$。杜威十进记号满足许多简单的数学性质,是对树进行分析的有用工具。这方面的一个例子是它给出任意一个树的节点的自然顺序的编号,这类似于本书内小节的顺序。2.3 节先于 2.3.1 小节,而且是在 2.2.6 小节之后。

在杜威十进记号和我们已经广泛使用的下标变量记号之间有密切的关系。如果 F 是一些树的森林,我们可以令 $F[1]$表示头一棵树的子树,使得 $F[1][2]\equiv F[1,2]$表示 $F[1]$的第二棵子树的子树,而 $F[1,2,1]$表示后者的头一个子森林,等等。在杜威十进记号下的节点 $a.b.c.d$ 是 $F[a,b,c,d]$的父亲。这个记号是通常的下标记号的一个扩充,因为每个下标可允许的范围依赖于前边的下标位置的值。

因此,特别地,我们看到,任何矩形数组都可以想像为一个树或森林结构的特殊情况。例如,下面是一个 3×4 矩阵的两种表示:

$$\begin{pmatrix} A[1,1] & A[1,2] & A[1,3] & A[1,4] \\ A[2,1] & A[2,2] & A[2,3] & A[2,4] \\ A[3,1] & A[3,2] & A[3,3] & A[3,4] \end{pmatrix}$$



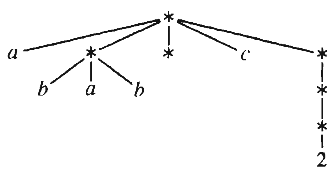
然而,重要的是要观察到,这个树结构不能忠实地反映出矩阵的所有结构,行关系明显地出现于树中,但列关系则不然。

森林进而又可看做通称为表结构的特殊情况。这里使用"表"(list)一词的非常技术性的意义,而且为了区分该词的技术性应用,我们将总是使用首字母大写的"List"(译作列表)。一个列表被(递归地)定义为*零个或多个原子或列表的有限序列*。这里"原子"是一个不明确的概念,指的是所需的任何对象的全域中的元素,只要有可能区分原子和列表即可。借助于包含逗号和括弧的明显的记号约定,我们可以区分原子和列表,并且可以方便地显示在一个列表内的顺序。作为一个例子,考虑

$$L=(a,(b,a,b),(\ ),c,(((2)))) \tag{3}$$

它是有 5 个元素的列表:头一个是原子 a,然后是列表(b,a,b),然后是空的列表$()$,然后原子 c,最后是列表$(((2)))$。后一列表由列表$((2))$组成,而它又由列表(2)组成,此列表则由原子 2 组成。

下列树结构对应于 L:



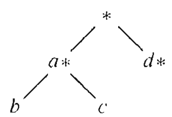
(4)

此图中的星号表示列表的定义与出现，它是相对于原子出现的。下标记号适用于列表，如同它适用于森林一样；例如，$L[2]=(b,a,b)$和 $L[2,2]=a$。

(4)中列表的节点除都是列表这一事实外，并没有引入什么数据。但是使用信息来标记列表的非原子的元素还是可能的，正如对于树和其它结构我们已经做过的那样；因此

$$A=(a:(b,c),d:(\ ))$$

将对应于可以如下画出的树：



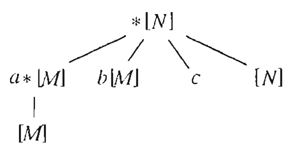
列表和树之间的最大区别在于列表可以重叠（即子列表不必是不相交的），而且甚至可以是递归的（可以包含本身），列表

$$M=(M) \tag{5}$$

对应于非树结构，列表

$$N=(a:M,b:M,c,N) \tag{6}$$

也一样。（在这些例子中，大写字母指的是列表，小写字母指的是标号和原子。）使用星号表示定义列表的每个位置，可以画出(5)和(6)的框图如下：



(7)

实际上，列表并不像上面的例子可能表示的那么复杂。实质上，它们是我们在 2.2 节中所考虑的线性表的简单推广，但附加上一个限制，即线性列表的元素可以是指向其它线性列表（而且可能指向它们本身）的链接变量。

小结：四种紧密相关的信息结构类型——树、森林、二叉树和列表——有许多来源，因此它们在计算机算法中是重要的。我们已经看到画出这些结构的框图的各种方法，我们还考虑了在谈及它们时有用的某些术语和记号。以下数节将对这些思想提供更多的细节。

习　题

1. [*18*] 有多少棵有 A，B 和 C 三个节点的不同树？

2. [*20*] 有多少棵有 A, B 和 C 三个节点的不同的有向树？

3. [*M20*] 从定义出发，严格地证明，在一棵树中，从每个节点 X 到根，有惟一的通路，即惟一的 $k\geq 1$ 的节点序列 $X_1, X_2, \cdots, X_k$，使得 X_1 是树的根，$X_k = X$，且对于 $1\leq j < k$，X_j 是 X_{j+1} 的父亲。(这个证明是关于树结构的几乎所有基本事实的典型证明。)提示：对树中的节点个数使用归纳法。

4. [*01*] 真或假：在一个传统的树的框图中(即根在顶部)，如果节点 X 比节点 Y 有更高的级，则在图中节点 X 低于节点 Y 而出现。

5. [*02*] 如果节点 A 有三个兄弟，而 B 是 A 的父亲，问 B 的度是多少？

▶ 6. [*21*] 通过与家谱图类比，如果 $m\geq 0$ 和 $n\geq 0$，试把命题"X 是 Y 相差了 n 代的第 m 个堂兄"定义为在一棵树中节点 X 和 Y 之间的一种有意义的关系。(关于家谱图的这些术语的意义，请查阅词典。)

7. [*23*] 对于所有 $m\geq -1$ 和所有整数 $n\geq -(m+1)$，这样来把上题中给出的定义推广，即对于一棵树的任何两个节点 X 和 Y，有惟一的 m 和 n，使得 X 是 Y 相差了 n 代的第 m 个堂兄。

▶ 8. [*03*] 什么二叉树不是树？

9. [*00*] 在(1)的两个二叉树中，哪一个节点是根(B 还是 A)？

10. [*M20*] 给定任何集合对 X, Y，如果或者 $X\subseteq Y$，或者 $X\supseteq Y$，或者 X 和 Y 不相交(换句话说，$X\cap Y$ 既可是 X, Y 也可是 $\emptyset$)，则说非空集合的一个汇集是嵌套的。图 20(a)指出任何树对应于嵌套集合的一个汇集；反之，每个这样的汇集都对应于一棵树吗？

▶ 11. [*HM32*] 通过习题 10 那样把树当做嵌套集合的汇集，而把树的定义推广到无穷树中。对于无穷树的每个节点，能否定义级、度、父亲和儿子这些概念？试给出对应于一棵树的实数的嵌套集合的例子，其中

a)每个节点有不可数的度和有无穷多个级；

b)有其级为不可数的节点；

c)每个节点的度至少为 2 和有不可数多个级。

12. [*M23*] 在什么条件下一个偏序集对应于一个无序的树或森林？(偏序集在 2.2.3 小节中定义。)

13. [*10*] 假设在杜威十进系统中，节点 X 被编号为 $a_1.a_2.\cdots.a_k$；问在从 X 到根的通路中诸节点的杜威号码是多少(参见习题 3)？

14. [*M22*] 设 S 是有形如"$1.a_1.\cdots.a_k$"的元素的任何非空集合，其中 $k\geq 0$ 和 $a_1, \cdots, a_k$ 是正整数。试证明当 S 有限而且满足下列条件时 S 确定一棵树："如果 $\alpha.m$ 在此集合中，则若 $m>1$，$\alpha.(m-1)$也在这个集合中，或若 $m=1$，α 也在这个集合中。"(对一棵树来说这个条件在杜威十进记号中显然被满足；因此它是表征树结构的另一种方式。)

▶ 15. [*20*] 类似于树节点的杜威十进记号，试设计出二叉树节点的一个记号。

16. [*20*] 画出类似于图 21 的对应于算术表达式(a) $2(a-b/c)$；(b) $a+b+5c$ 的树。

17. [*01*] 如果 Z 代表当做森林的图 19，什么节点是($Z[1,2,2]$)的父亲？

18. [*08*] 在列表(3)中，$L[5,1,1]$是什么？$L[3,1]$又是什么？

19. [*15*] 对于列表 $L=(a,(L))$，画出类似于(7)的一个列表框图？在这个列表中 $L[2]$是什么？$L[2,1,1]$又是什么？

▶ 20. [*M21*] 试把一棵 0-2 树定义为每个节点恰有零个或两个孩子的树。(形式地说，一棵 0-2 树由一个称做它的根的节点，加上 0 个或 2 个不相交的 0-2 树组成。)证明每棵 0-2 树有奇数个节点；并给出有 n 个节点的二叉树和有 $2n+1$ 个节点的(有序)0-2 树之间的一一对应。

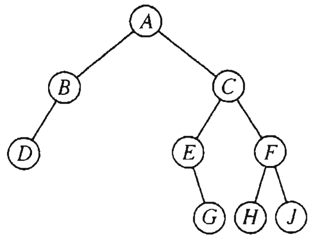
21. [*M22*] 如果一棵树有度数 1 的 n_1 个节点,度数 2 的 n_2 个节点,…,度数 m 的 n_m 个节点,则它有多少个终端节点?

▶ 22. [*21*] 标准的欧洲纸张的大小 A0,A1,A2,…,An,…是长宽比为$\sqrt{2}$比 1 的矩形,因此它们的面积是 2^{-n} m^2。所以,如果我们把 An 的纸裁成两半,我们就得到两张 A$(n+1)$的纸。试利用这一原理设计二叉树的图形表示,并通过画出下面的 2.3.1-(1)的表示来说明你的想法。

2.3.1　遍历二叉树

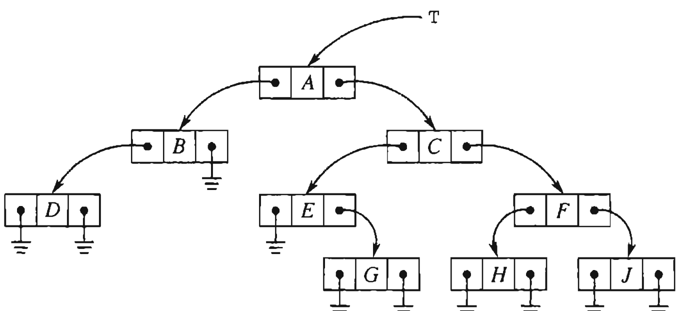
在对树进一步研究之前,重要的是要很好地理解二叉树的性质,因为在计算机内一般的树是借助于某个等价的二叉树来表示的。

我们已经把二叉树定义为一个有限的节点集合,它或者为空,或者由一个根连同两个二叉树组成。这个定义提示了在一台计算机内表示二叉树的自然的方式:在每个节点内,我们可以有两个链接,LLINK 和 RLINK 以及作为"指向树的指针"的链接变量 T。如果这棵树为空,T = Λ;否则 T 是树的根节点的地址,而 LLINK(T)和 RLINK(T)分别是指向根的左子树和右子树的指针。这些规则递归地定义任何二叉树的内存表示;例如,



(1)

通过



(2)

来表示。

这个简单自然的表示说明了二叉树结构的特别重要性。我们将在 2.3.2 小节看到,一般的树可以方便地表示为二叉树。而且,出现在一些应用中的许多树本身固有地就是二叉的,因此二叉树本身就是令人感兴趣的。

对于树结构的操作有许多算法,而且反复地出现于这些算法中的是*遍历*或者"走遍"一棵树的概念。这是系统地考虑树的节点,使得每个节点恰被访问一次的方法。对一棵树完整的遍历给出节点的一种线性安排,而且如果我们能够在这样的顺序之下谈

及一个给定节点后边或前边的“下一个”节点，就会使许多算法变得容易。

有三个主要的方法可以用来遍历二叉树：可以以先根序(preorder)，中根序(inorder)或后根序(postorder)来访问诸节点。这三种方法是递归地定义的。当二叉树为空时，什么都不做即可“遍历”；否则遍历分三步进行。

先根序	中根序	后根序
访问根	遍历左子树	遍历左子树
遍历左子树	访问根	遍历右子树
遍历右子树	遍历右子树	访问根

如果将这些定义应用到(1)和(2)的二叉树，我们发现在先根序下的节点是

$$A \quad B \quad D \quad C \quad E \quad G \quad F \quad H \quad J \tag{3}$$

(首先根 A 出现，然后在先根序下出现的是左子树



最后我们以先根序遍历右子树。)对于中根序，我们在访问左子树和右子树的节点之间访问根，实际上就好像把这些节点投影到一条水平直线上一样，而这就给出顺序

$$D \quad B \quad A \quad E \quad G \quad C \quad H \quad F \quad J \tag{4}$$

类似地，对于这个二叉树的节点的后根序是

$$D \quad B \quad G \quad E \quad H \quad J \quad F \quad C \quad A \tag{5}$$

我们将看到把二叉树的节点安排成一个序列的这三个方法是极端重要的，因为它们同处理树的大多数计算机方法密切相关。当然先根序，中根序和后根序这些名称来自于根相对于它的子树的相对位置。在二叉树的许多应用中，在左子树和右子树的意义之间存在有对称性，因此在这样的情况下：对称序这一术语可作为中根序的同义词。中根序把根放在中间，实际上在左和右之间是对称的：如果相对于垂直轴把二叉树反射，则对称序只不过颠倒顺序而已。

为了可直接地应用于计算机实现上，对递归地叙述的定义，例如对于三个基本的顺序刚刚给出的定义，必须重新给出。在第 8 章将讨论其一般方法；和下面的算法一样，我们通常利用一个辅助栈。

算法 T（*以中根序遍历二叉树*） 设 T 是有(2)那样的表示的指向二叉树的指针；本算法以中根序访问二叉树中的所有节点，并且利用一个辅助栈 A。

T1.［初始化］ 置栈 A 为空，并置链接变量 P←T。

T2.［P = Λ?］ 如果 P = Λ，转到步骤 T4。

T3.［栈⇐P］（现在 P 指向要加以遍历的一个非空二叉树。）置 A⇐P；即是，把 P 的值放入栈 A。(见 2.2.1 小节。)然后置 P←LLINK(P)并返回步骤 T2。

T4.［P⇐栈］ 如果栈 A 为空，则算法终止；否则置 P⇐A。

T5.［访问 P］访问 NODE(P)，然后置 P←RLINK(P)，并返回步骤 T2。 ▎

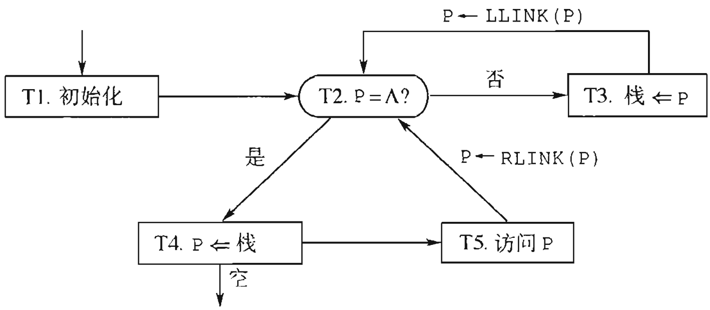


图 23　用于中根序遍历的算法 T

在这个算法的最后一步里，"访问"一词的意思是当遍历该树时我们打算进行的任何活动。相对于这其它的活动，算法 T 像一个共行程序那样运行：每当主程序要 P 从一个节点移动到它在中根序下的后继时主程序就启动这个共行程序。当然，由于这个共行程序仅仅在一处调用主程序，因此它同一个子程序没有大的区别（见 1.4.2 小节）。算法 T 假定，外部活动既不从树删除 NODE(P)，也不删除它的任何祖宗。

读者现在应该利用二叉树(2)作为一个测试情况来尝试运行算法 T，以便了解这个过程背后的原因。当我们达到步骤 T3 时，我们要遍历根是由指针 P 指出的二叉树。想法是把 P 存在栈中而后遍历左子树；当做完这之后，我们将达到步骤 T4 并且将再次在栈中找到 P 的旧值。在步骤 T5 访问了根 NODE(P)之后，剩下的工作就是遍历右子树。

算法 T 在稍后的其它许多算法中是典型的，因此，考察对于上一段落所作说明的一个形式证明是有教益的。现在让我们通过对 n 使用归纳法，尝试证明算法 T 以中根序遍历 n 个节点的二叉树。如果我们能证明稍微更一般的结果，那我们的目标就很容易实现了。

以 P 作为指向 n 个节点的二叉树的指针，且对于某个 $m \geq 0$，有包含 A[1]…A[m]的栈 A，从步骤 T2 开始，步骤 T2～T5 的过程将以中根序遍历问题中的二叉树，而且将以栈 A 返回到它原来的值 A[1]…A[m]达到步骤 T4。

由于步骤 T2，当 $n = 0$ 时这个命题显然是正确的。如果 $n > 0$，令 P_0 是在进入步骤 T2 时 P 的值，由于 $P_0 \neq \Lambda$，我们将执行步骤 T3，它意味着栈 A 被修改成 A[1]…A[m]P_0 以及 P 被置为 LLINK(P_0)。现在左子树有比 n 少的节点，所以由归纳法我们将以中根序遍历左子树并且最终以 A[1]…A[m]P_0 在栈中而到达步骤 T4。步骤 T4 把栈恢复成 A[1]…A[m]并且置 P←P_0。步骤 T5 现在访问 NODE(P_0)并置 P←RLINK(P_0)。现在右子树节点少于 n，所以由归纳法我们将以中根序遍历右子树而且按要求到达步骤 T4。按照该顺序的定义，该树已被以中根序遍历。这就完成了证明。

可以用几乎完全相同的算法来叙述以先根序遍历二叉树（见习题 12）。以后根序实现遍历稍微困难些（见习题 13），由于这个原因，后根序对于二叉树来说就不像其它两个那样重要。

在这些不同顺序下定义节点的后继和前驱的新记号就很方便了。如果 P 指向一个二叉树的节点,令

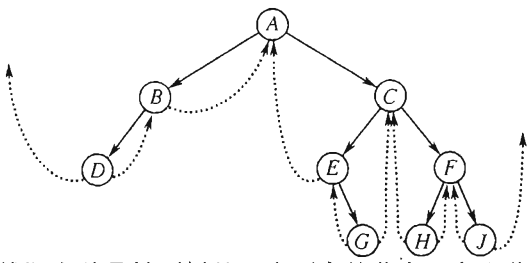
$$
\begin{aligned}
&P* = \text{在先根序下 NODE(P)的后继的地址;}\\
&P\$ = \text{在中根序下 NODE(P)的后继的地址;}\\
&P\# = \text{在后根序下 NODE(P)的后继的地址;}\\
&*P = \text{在先根序下 NODE(P)的前驱的地址;}\\
&\$P = \text{在中根序下 NODE(P)的前驱的地址;}\\
&\#P = \text{在后根序下 NODE(P)的前驱的地址}
\end{aligned}
\qquad (6)
$$

如果 NODE(P)没有这样的后继或前驱,一般使用 LOC(T)的值,其中 T 是问题中的树的外部指针。我们有 *(P*)=(*P)* = P, $(P$)=($P)$ = P 以及 #(P#)=(#P)# = P。作为这种表示法的一个例子,令 INFO(P)是在树(2)中的 NODE(P)所示的字母;则如果 P 指向这个根,我们有 INFO(P)= A,INFO(P*)= B,INFO(P$)= E,INFO($P) = B,INFO(#P)= C,以及 P# = *P = LOC(T)。

这时,读者可能会感觉到对于 P*,P$ 等的直观意义的不安全性。随着我们继续进行下去,这些思想将逐渐地变得清晰;这一小节的习题 16 也可能是有帮助的。在“P$”中的“$”旨在提示字母 S,表示“对称(symmetric)序”。

(2)中给出的二叉树的内存表示还有一个重要的变体,它有点类似于循环表和直线单向表之间的区别。注意在树(2)中,空链接要比其它指针多,而且用传统方法表示任何二叉树确实如此(见习题 14)。但我们并不真的需要浪费所有这些内存空间。例如,对于每个节点我们可以保存两个“标志”指示符,它只是以内存的两位来告知 LLINK 或 RLINK 或两者为空;于是终端链接的内存空间就可留作它用。

A. J. Perlis 和 C. Thornton 曾提出使用额外空间的巧妙方法,他们设计了所谓的穿线的树表示。在这个方法中,终端链接作为对遍历的辅助,被穿到树其它部分的“线”所代替。等价于(2)的穿线树是



(7)

这里的虚线表示“穿线”,它总是转到树的一个更高的节点。每个节点现在有两个链接:某些节点,好比 C,有通常的对左子树和右子树的链接;其它节点,比如 H,有两个穿线的链接;而某些节点有每种类型的一个链接。从 D 和 J 发出的特殊穿线将在稍后说明。它们出现在“最左”和“最右”节点中。

在一个穿线的二叉树的内存表示中,有必要对虚线和实线加以区别。这可以像上边所建议的那样,通过在每个节点中的两个附加的一个二进位字段,即 LTAG 和 RTAG 来完成。穿线表示可以精确地定义如下:

无穿线表示	穿线表示
LLINK(P) = Λ	LTAG(P) = 1， LLINK(P) = $P
LLINK(P) = Q≠Λ	LTAG(P) = 0， LLINK(P) = Q
RLINK(P) = Λ	RTAG(P) = 1， RLINK(P) = P$
RLINK(P) = Q≠Λ	RTAG(P) = 0， RLINK(P) = Q

按照这个定义，每个新的穿线链接直接以对称序（中根序）指向问题中的这个节点的前驱或后继。图 24 说明在任何二叉树中穿线链接的一般方向。

在某些算法中，可以保证，任何子树的根在内存中出现的位置总比该子树的其它节点出现的位置要低些。于是当且仅当 LLINK(P) < P 时，LTAG(P)将为 1，所以 LTAG 将是多余的。由于同样的理由 RTAG 也将是多余的。

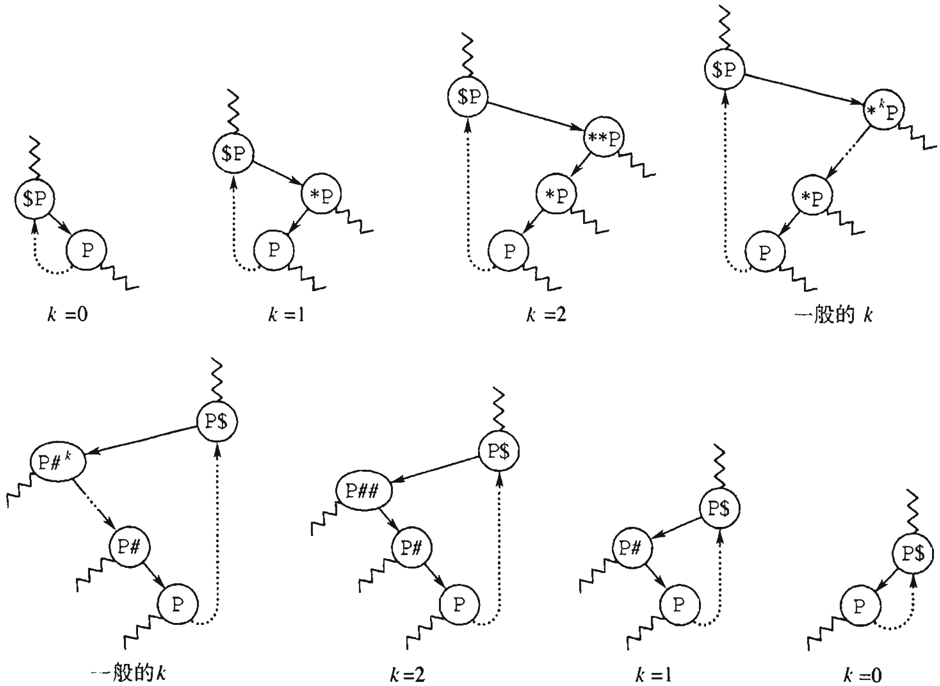


图 24　在一个穿线二叉树中左和右穿线链接的一般方向。
波浪线表示对树其它部分的链接或穿线

穿线树的重大优点是遍历算法变得较简单了。例如，给定 P，以下算法计算P$：

算法 S（在穿线二叉树中的对称序（中根序）后继）　如果 P 指向穿线二叉树的一个节点，这个算法置 Q←P$。

S1.［RLINK(P)是一个穿线吗？］置 Q←RLINK(P)。如果 RTAG(P) = 1，终止此算法。

S2. [向左搜索] 如果 LTAG(Q) = 0,置 Q←LLINK(Q),并重复这一步骤。否则算法终止。 ▌

注意,这里无需使用栈来实现在算法 T 中使用栈所做的工作。事实上,通常的表示(2),如果仅仅给定在树中的一个随机指针 P 的地址,将造成不可能有效地来找出 P$。由于在一个无穿线的表示中没有向上指的链接,因此对于在一个给定点上面的是什么节点,没有提供任何线索,除非我们保留如何抵达这一点的历史。当没有穿线时,算法 T 中的栈就提供了必要的历史。

我们断言,算法 S 是“有效的”,尽管这个性质并不是立即明了的,因为步骤 S2 可执行任意次。就步骤 S2 中的循环而言,如同算法 T 那样使用栈也许会更快些?为了研究这个问题,我们将考虑在 P 是树的一个“随机”点时步骤 S2 必须执行的平均次数;或者与此相同地,如果反复地使用算法 S 来遍历整棵树,我们将确定步骤 S2 执行的总次数。

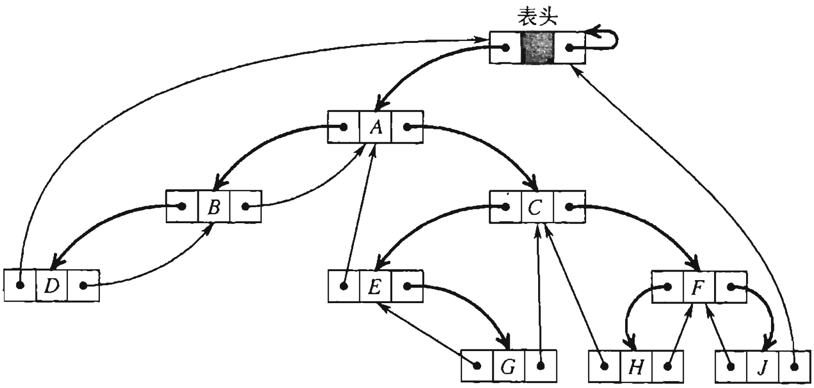
在进行这个分析的同时,研究算法 S 和 T 两者完整的程序将是有教益的。和通常一样,我们应当仔细地建立所有算法以使这些算法能正确地处理空的二叉树;而且如果 T 是指向这个树的指针,我们将希望 LOC(T)* 和 LOC(T)$ 分别是在先根序或对称序下的头一个节点。对于穿线树,如果把 NODE(LOC(T))做成树的“表头”,而且

$$\begin{aligned}&\text{LLINK(HEAD)} = \text{T}, &&\text{LTAG(HEAD)} = 0\\ &\text{RLINK(HEAD)} = \text{HEAD}, &&\text{RTAG(HEAD)} = 0\end{aligned} \qquad (8)$$

结果事情将会做得很好。(这里 HEAD 表示 LOC(T),即表头的地址。)一个空的穿线树将满足下列条件:

$$\text{LLINK(HEAD)} = \text{HEAD}, \qquad \text{LTAG(HEAD)} = 1 \qquad (9)$$

树通过把节点插到表头的*左边*而增长。(这些初始条件主要由计算 P* 的算法指定,它出现于习题 17 中。)按照这些约定,作为一棵穿线树,二叉树(1)的计算机表示是



(10)

通过以上预备措施,我们可以接着考虑算法 S 和 T 的 MIX 版本。以下的程序假定二叉树的节点有双字的形式

LTAG	LLINK	INFO1
RTAG	RLINK	INFO2

在无穿线树中，LTAG 和 RTAG 将总是"+"而终端链接将通过零来表示。在穿线树中，我们将使用"+"来代表为 0 的标志而用"-"来代表为 1 的标志。缩写 LLINKT 和 RLINKT 将分别用来代表组合的 LTAG - LLINK 和 RTAG - RLINK 字段。

两个标志位占据 MIX 字的原来未加使用的符号位置，因此并未浪费内存空间。类似地，对于 MMIX 计算机我们也将有能力像"免费地"使用标志位那样使用链接字段的最低有效位，因为指针的值一般地将是偶数，而且还因为 MMIX 将使在对内存编址时可容易地忽略低有效位。

以下两个程序以对称序(即中根序)遍历二叉树，通过变址寄存器 5 指向当前感兴趣的节点时，它们周期性地跳转到单元 VISIT。

程序 T　在算法 T 的这个实现中，栈保存在单元 A+1，A+2，…，A+MAX 中；rI6 是栈指针且 rI5≡P。如果栈增长得太大，则 OVERFLOW 出现。这个程序已经对算法 T 稍微作了重新安排(步骤 T2 出现三次)，使得当从 T3 直接转到 T2 再到 T4 时，不需要检查空栈。

```
LLINK   EQU    1:2
RLINK   EQU    1:2
T1      LD5    HEAD(LLINK)    1    T1. 初始化。置 P←T
T2A     J5Z    DONE           1    如果 P=Λ,停止
        ENT6   0              1
T3      DEC6   MAX            n    T3. 栈⇐P
        J6NN   OVERFLOW       n    栈已达到容量了吗?
        INC6   MAX+1          n    如果没有,栈指针增 1
        ST5    A,6            n    把 P 存入栈中
        LD5    0,5(LLINK)     n    P←LLINK(P)
T2B     J5NZ   T3             n    如果 P≠Λ,转到 T3
T4      LD5    A,6            n    T4. P⇐栈
        DEC6   1              n    栈指针减 1
T5      JMP    VISIT          n    T5. 访问 P
        LD5    1,5(RLINK)     n    P←RLINK(P)
T2C     J5NZ   T3             n    T2 .P=Λ?
        J6NZ   T4             a    检测栈是否为空
DONE    ...                        ▌
```

程序 S　本程序扩充了算法 S，增加了初始条件和结果条件，使得这个程序可同程序 T 相对照。

```
LLINKT  EQU    0:2
RLINKT  EQU    0:2
S0      ENT5   HEAD             1       S0. 初始化。置 P←HEAD
        JMP    2F               1
S3      JMP    VISIT            n       S3. 访问 P
S1      LD5N   1,5(RLINKT)      n       S1. RLINK(P)是一个穿线吗?
        J5NN   1F               n       如果 RTAG(P) = 1 则转移
        ENN6   0,5              n - a   否则置 Q←RLINK(P)
S2      ENT5   0,6              n       S2. 向左找。置 P←Q
2H      LD6    0,5(LLINKT)      n + 1   Q←LLINK(P)
        JP6    S2               n + 1   如果 LTAG(P) = 0,重复
1H      ENT6   -HEAD,5          n + 1
        J6NZ   S3               n + 1   除非 P = HEAD,否则访问  ▌
```

对运行时间的分析和上面的代码一起出现。使用基尔霍夫定律和以下事实:

i) 在程序 T 中,插入栈中的个数必须等于删去的个数。

ii) 在程序 S 中,对每个节点的 LLINK 和 RLINK 恰考察一次。

iii) "访问"的次数是树中的节点数。

这个分析告诉我们,程序 T 花费 $15n + a + 4$ 个时间单位,程序 S 花费 $11n - a + 7$ 个时间单位,其中 n 是树中节点的个数,而 a 是终端右链接(即无右子树的节点)的个数。假定 $n \neq 0$,a 可以低至 1,也可以高至 n。如果左和右是对称的,作为在习题 14 中证明的事实的结果,a 的平均值为 $(n+1)/2$。

以这个分析为基础,我们可以获得的主要结论为:

i) 如果 P 是树的一个随机节点,则对于算法 S 的每次执行,平均说来,步骤 S2 仅执行一次。

ii)对穿线树来说,遍历是稍微快的,因为它不需要栈操作。

iii) 由于需要辅助栈,算法 T 比算法 S 需要更多的存储空间。在程序 T 中,我们在连续的存储单元中保存栈;因此需要对它的大小设置一个任意的界。如果超过了这个界,那将是非常难堪的,因此必须把它设置得相当大(见习题 10);因此程序 T 的内存要求要比程序 S 大得多。一个复杂的计算机应用并非不经常同时独立地遍历若干树,因此在程序 T 中对于每棵树将需要分开的栈。这就提示,程序 T 可以对它的栈使用链接分配(请见习题 20);于是尽管当把其它共行程序的执行时间加进来时,遍历的速度并不非常重要,但它的执行时间变成 $30n + a + 4$,粗略地是以前的两倍。还有另一个选择,如同在习题 21 中所讨论那样,是把栈的链接保持在树本身之内。

iv) 当然,算法 S 要比算法 T 更为一般,因为当不必遍历整个二叉树时,它允许我们从 P 转到 P$ 。

因此相对于遍历而言,穿线二叉树肯定地要比无穿线二叉树优越。但这些优点在某些应用中被在一个穿线树中插入和删除节点所需时间的稍微增加所抵消。有时有可

能通过与无穿线表示“共享”公共子树来节省内存空间，然而穿线树要求符合严格的树结构，其中不能有重叠的子树。

穿线链接可以用于计算 P＊，$P 和 #P，其效率可与算法 S 相当。函数＊P 和 P# 稍微难于计算，如同它们对于非穿线树表示那样。读者最好做一做习题 17。

如果在一开头就很难建立穿线链接，则穿线树的大多数用途就会消失。使得这个想法真正有效的是穿线树几乎就像通常的树那样容易增长。我们有下列算法：

算法 I（插入到穿线二叉树中） 如果右子树为空（即如果 RTAG(P) = 1），这个算法就把节点 NODE(Q)作为 NODE(P)的右子树附加上来；否则就把 NODE(Q)插到 NODE(P)和 NODE(RLINK(P))之间，使得后一节点成为 NODE(Q)的右子节点。假定发生插入的二叉树是如同(10)中那样穿线的；习题 23 中给出另一种情况。

I1.［调整标志和链接］置 RLINK(Q)←RLINK(P)，RTAG(Q)←RTAG(P)，RLINK(P)←Q，RTAG(P)←0，LLINK(Q)←P，LTAG(Q)←1。

I2.［RLINK(P)是穿线吗?］如果 RTAG(Q) = 0，置 LLINK(Q$)←Q。（这里 Q$ 由算法 S 确定，即使 LLINK(Q$)现在指向 NODE(P)而不是 NODE(Q)，它仍将正确地工作。仅当插入到穿线树的中间而不仅仅是插入新叶时这个步骤才必要。）▌

通过把左和右的作用颠倒（特别是，通过在步骤 I2 中以 $Q 来代替 Q$），我们得到以类似方式插入左边的一个算法。

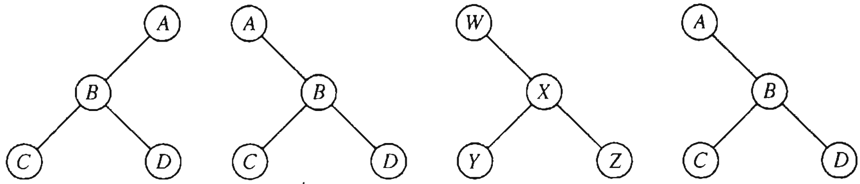
迄今为止我们关于穿线二叉树的讨论既利用了左边的穿线链接也利用了右边的穿线链接。在完全无穿线的表示方法和完全穿线的表示方法之间有一个重要的中间地带：通过利用穿线的 RLINK，同时以 LLINK = Λ 表示空的左子树，*右穿线二叉树*把这两个方法组合在一起。（类似地，左穿线二叉树仅穿线空的诸 LLINK。）算法 S 并不是实际地利用穿线的 LLINK；如果我们把步骤 S2 中的测试“LTAG = 0”改变为测试“LLINK ≠ Λ”，就得到以对称序遍历右穿线二叉树的算法。在右穿线情况下不用作任何变动，程序 S 就有效。大量二叉树结构的应用只要求使用函数 P$ 和/或 P＊来进行对树的一个自左至右的遍历，而且对于这些应用来说，不需要对 LLINK 穿线。我们已经在左的方向和右的方向两方面描述了穿线方法，以便指出该种状况的对称性及诸可能性。但在实践中，单边的穿线要普遍得多。

现在让我们考虑二叉树的一个重要性质以及它同遍历的联系。两个二叉树 T 和 T'，如果它们结构相同，便说它们是*相似的*；形式地说，这意味着，(a)它们两者都为空，或者(b)它们都是非空的而且它们两者的左和右子树分别地相似。非形式地说，相似性意味着，T 和 T' 的框图有相同的“形状”。表述相似性的另一个方法是说，存在保持结构的 T 和 T' 的节点之间的一一对应：如果在 T 中的节点 u_1 和 u_2 分别对应于 T' 中的 u'_1 和 u'_2，则当且仅当 u'_1 处在 u'_2 的左子树中时 u_1 也处于 u_2 的左子树中，而且对于右子树同样的结论也成立。

如果二叉树 T 和 T' 相似而且对应的节点包含相同的信息，则说它们是*等价的*。形式地说，令 info(u)表示包含在一个节点 u 中的信息；树等价当且仅当(a)它们都为空，或者(b)它们两者都非空，而且 info(root(T)) = info(root(T'))，而且它们的左和右子树

分别地也等价。

作为这些定义的一个例子,考虑四个二叉树



其中,头两个不相似,第二、第三和第四个相似,第二和第四个等价。

涉及树结构的某些计算机应用需要有一种算法来判定两个二叉树是否相似或等价。在这方面,下列定理是有用的:

定理 A *令二叉树 T 和 T' 的节点在先根序下分别为*

$$u_1, u_2, \cdots, u_n \quad 和 \quad u'_1, u'_2, \cdots, u'_{n'}$$

对于任何节点 u,令

$$\begin{aligned}&如果\ u\ 有一个非空左子树,则\ l(u) = 1,\ 否则\ l(u) = 0;\\&如果\ u\ 有一个非空右子树,则\ r(u) = 1,否则\ r(u) = 0\end{aligned} \tag{11}$$

则 T 和 T' 相似当且仅当 $n = n'$,而且

$$对于\ 1 \leqslant j \leqslant n, l(u_j) = l(u'_j), r(u_j) = r(u'_j) \tag{12}$$

而且,当且仅当附加条件

$$对于\ 1 \leqslant j \leqslant n, \mathrm{info}(u_j) = \mathrm{info}(u'_j) \tag{13}$$

成立时,T 和 T' 才是等价的。

注意 l 和 r 是穿线树中 LTAG 和 RTAG 二进位的反码。这个定理借助于 0 和 1 的两个序列来表征任何二叉树的结构。

证明 显然,如果我们证明了相似性的条件,就会立即得出二叉树等价的条件;其次,$n = n'$ 和(12)的条件肯定是必要的,因为相似树的对应节点在先根序下有相同的位置。其次,只要证明条件(12)和 $n = n'$ 是保证 T 和 T' 相似的充分条件就足够了。使用以下的辅助结果,对 n 使用归纳法即可证明。

引理 P *设非空二叉树的节点在先根序下是 $u_1, u_2, \cdots, u_n$,并设 $f(u) = l(u) + r(u) - 1$。则*

$$f(u_1) + f(u_2) + \cdots + f(u_n) = -1 \quad 且 \quad f(u_1) + \cdots + f(u_k) \geqslant 0, 1 \leqslant k < n \tag{14}$$

证明　对于 $n=1$，这个结果是显然的。如果 $n>1$，二叉树由它的根 u_1 和进一步的节点组成。如果 $f(u_1)=0$，则或者是左子树，或者是右子树为空，所以由归纳法，条件显然为真。如果 $f(u_1)=1$，令左子树有 n_l 个节点；由归纳法，我们有

$$\text{对于 } 1 \leqslant k \leqslant n_l, f(u_1)+\cdots+f(u_k)>0; f(u_1)+\cdots+f(u_{n_l+1})=0 \qquad (15)$$

因而条件(14)再次是显然的。▌

(关于类似于引理 P 的其它定理，请参看第 10 章关于波兰记号的讨论。)

为了完成定理 A 的证明，我们注意到当 $n=0$ 时定理显然成立。如果 $n>0$，先根序的定义意味着 u_1 和 u'_1 分别是它们的树的根，而且存在整数 n_l 和 n'_l(左子树的大小)使得

$$u_2,\cdots,u_{n_l+1} \text{ 和 } u'_2,\cdots,u'_{n'_l+1} \text{ 是 } T \text{ 和 } T' \text{ 的左子树；}$$

$$u_{n_l+2},\cdots,u_n \text{ 和 } u'_{n'_l+2},\cdots,u'_n \text{ 是 } T \text{ 和 } T' \text{ 的右子树。}$$

如果我们能证明 $n_l=n'_l$，归纳法证明就将完成。有三种情况：

如果 $l(u_1)=0$，则 $n_l=0=n'_l$；

如果 $l(u_1)=1, r(u_1)=0$，则 $n_l=n-1=n'_l$；

如果 $l(u_1)=r(u_1)=1$，则由引理 P 我们可以找到最小的 $k>0$，使得 $f(u_1)+\cdots+f(u_k)=0$；而且 $n_l=k-1=n'_l$(见(15))。▌

作为定理 A 的一个结果，我们可以通过以先根序遍历两个穿线二叉树，并检查 INFO 和 TAG 字段，来检查这两个树的等价性和相似性。A. J. Blikle 已经得到定理 A 的某些有趣推广，*Bull. de l'Acad. Polonaise des Sciences*, Série des Sciences Math., Astr. Phys. **14** (1966), 203 ~ 208；他考虑了可能的遍历顺序的一个无穷类别，其中仅六个(包含先根序)由于它们简单性质而被称做“无地址的”。

我们通过给出一个典型的，而对二叉树来说又是基本的算法来结束这一小节，这个算法把一个二叉树复制到不同的内存单元中。

算法 C(**复制一个二叉树**)　设 HEAD 是一个二叉树 T 的表头地址。于是，T 是通过 LLINK(HEAD)抵达的 HEAD 的左子树。令 NODE(U)是带有空左子树的一个节点。这个算法给出 T 的一个副本而且这个副本变成 NODE(U)的左子树。特别是，如果 NODE(U)是一个空的二叉树的表头，则这个算法把空树变成为 T 的一个副本。

C1. [初始化]　置 P←HEAD，Q←U。转到 C4。

C2. [右边有任何东西?]　如果 NODE(P)有一个非空的右子树，置 R⇐AVAIL，并把 NODE(R)附加到 NODE(Q)的右边。(在步骤 C2 开始时，NODE(Q)的右子树为空。)

C3. [复制 INFO]　置 INFO(Q)←INFO(P)。(这里 INFO 表示除开链接之外，要加以复制的节点的所有部分。)

C4. [左边有任何东西?]　如果 NODE(P)有一个非空的左子树，置 R⇐AVAIL，并把

NODE(R)附加到 NODE(Q)的左边。(在步骤 C4 开始时,NODE(Q)的左子树为空。)

C5.[前推] 置 P←P*,Q←Q*。

C6.[检查完成否] 如果 P=HEAD(或等价地如果 Q=RLINK(U),假定 NODE(U)有一个非空的右子树),算法结束;否则转到步骤 C2。 ▎

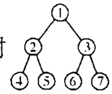
这个简单的算法显示了树遍历的典型应用。这里的描述适用于穿线的、无穿线的或部分穿线的树。步骤 C5 要求对先根序的后继 P* 和 Q* 的计算;对于无穿线的树,一般通过一个辅助的栈完成。算法 C 的正确性的证明出现于习题 29 中;在右穿线二叉树的情况下对应于这个算法的一个 MIX 程序出现于习题 2.3.2-13 中。对于穿线树,在步骤 C2 和 C4 中的"附加"是使用算法 I 来完成的。

以下习题包括了同本小节的材料有关的不少有趣课题。

Binary or dichotomous systems, although regulated by a principle,
are among the most artificial arrangements
that have ever been invented.
二叉或两分支的系统,尽管有其自身规则,
是属于人们所发明的最人为的安排之一。
——WILLIAM SWAINSON, *A Treatise on the Geography and Classification of Animals* (1835)

习　题

1.[*01*] 令 INFO(P)表示(2)中保存于 NODE(P)中的字母。INFO(LLINK(RLINK(RLINK(T))))是什么?

2.[*11*] 以(a)先根序;(b)对称序;(c)后序列出二叉树  的节点。

3.[*20*] 下列命题是对还是错:"在先根序,中根序和后根序中二叉树的终节点出现在相同的相对位置。"

▶ 4.[*20*] 书中正文定义了遍历二叉树的三种基本的顺序;另一个选择以如下三步来进行:

a)访问根,

b)遍历右子树,

c)遍历左子树,

对所有非空子树递归地使用同一个规则。这个新的顺序与已经讨论过的三种顺序是否有任何简单的关系?

5.[*22*] 在类似于对于树的"杜威十进记号"的记号下,通过一系列的 0 和 1,可以标识一个二叉树的节点:根(如果存在的话)以序列"1"来表示,由 α 表示的节点的左子树和右子树的根(如果存在的话)分别由 $\alpha0$ 和 $\alpha1$ 表示,例如,在(1)中的节点 H 将表示为"1110"。(见习题 2.3-15。)

证明借助于这个记号可方便地描述先根序，中根序和后根序。

6.［*M22*］假设一个二叉树有 n 个节点，在先根序下它们是 $u_1 u_2 \cdots u_n$，而在中根序下它们是 $u_{p_1} u_{p_2} \cdots u_{p_n}$，证明通过在习题 2.2.1-2 意义下把 1 2⋯ n 传送到一个栈中，可以得到排列 $p_1\ p_2 \cdots p_n$。反过来，证明，以这样的方法，通过对应于某个二叉树的栈可得到任何排列 $p_1\ p_2 \cdots p_n$。

7.［*22*］证明如果给出了一个二叉树的节点的先根序和中根序，就可以构造出二叉树的结构。如果给了我们先根序和后根序（而不是中根序），同一结果是否仍然保持为真？而如果给了我们中根序和后根序又如何？

8.［*20*］找出其节点在(a)先根序和中根序；(b)先根序和后根序；(c)中根序和后根序各两种顺序下，恰好以相同序列出现的所有二叉树。

9.［*M20*］当使用算法 T 遍历有 n 个节点的一个二叉树时，指出步骤 T1，T2，T3，T4 和 T5 各被执行多少次（以 n 的函数给出）。

▶ 10.［*20*］在执行算法 T 期间，如果二叉树有 n 个节点，则一次可放进栈中的最大项数是多少？（这个问题的答案对于存储分配是非常重要的，如果栈连续存储的话。）

11.［*HM41*］假定有 n 个节点的所有二叉树被认为是概率相等的，作为 n 的函数，试分析在执行算法 T 期间出现的最大栈大小的平均值。

12.［*22*］试设计类似于算法 T 的一个算法，它以*先根序*遍历一个二叉树，并证明你的算法是正确的。

▶ 13.［*24*］试设计类似于算法 T 的一个算法，它以*后根序*遍历一个二叉树。

14.［*22*］证明如果带有 n 个节点的一个二叉树像在(2)中那样表示，在这个表示中 Λ 链接的总个数可表达为 n 的一个简单函数；这个量不依赖于树的形状。

15.［*15*］在像(10)那样的一个穿线树表示中，每个节点，表头除外，都恰有一个从上面指向它的链接，即是从它的父亲来的链接。某些节点也有从下面指向它们的链接；例如，包含 C 的节点有从下面出来的两个指针，而节点 E 恰有一个。在指向一个节点的链接数和该节点的某个其它基本性质之间是否有什么简单联系？（我们需要知道，当改变树的结构时，需要多少链接指向一个给定的节点。）

▶ 16.［*22*］图 24 中的框图借助于靠近 NODE(Q)的结构，帮助提供在一个二叉树中 NODE(Q$)的位置的一个直观特征：如果 NODE(Q)有一个非空的右子树，在上面的框图中，考虑 Q=$P 和 Q$ = P；NODE(Q$)是该右子树的“最左”节点。如果 NODE(Q)有一个空的右子树，在下面的那个框图中，考虑 Q = P；通过在树中往上走，直到做完了通向右边的头一个向上的步骤为止，可找到 NODE(Q$)。

借助于靠近 NODE(Q)的结构，给出用于寻找在一个二叉树中 NODE(Q＊)的位置的类似的“直观”的规则。

▶ 17.［*22*］给出一个用于确定在一个穿线二叉树中的 P＊的类似于算法 S 的算法。假定树有像(8)，(9)和(10)那样的表头。

18.［*24*］使用我们可以称之为*双重序*的先根序和中根序的组合，许多处理树的算法喜欢访问每个节点*两次*而不是一次。在双重序下遍历一个二叉树定义如下：如果二叉树是空的，则什么也不做；否则

a)头一次访问根；

b)以双重序遍历左子树；

c)第二次访问根；

d)以双重序遍历右子树。

例如，在双重序下遍历(1)给出序列

$$A_1 B_1 D_1 D_2 B_2 A_2 C_1 E_1 E_2 G_1 G_2 C_2 F_1 H_1 H_2 F_2 J_1 J_2$$

其中 A_1 表示 A 头一次被访问。

如果 P 指向树的一个节点而且 $d=1$ 或 2，若在访问了 NODE(P)第 d 次之后在双重序下下一步是第 e 次访问 NODE(Q)，则定义$(\text{P},d)^{\Delta}=(\text{Q},e)$；或者，如果在双重序下$(\text{P},d)$是最后一步，我们写$(\text{P},d)^{\Delta}=(\text{HEAD},2)$，其中 HEAD 是表头的地址。我们也定义$(\text{HEAD},1)^{\Delta}$作为在双重序下的头一步。

试设计一个在双重序下遍历一个二叉树的类似于算法 T 的算法，而且还设计一个计算$(\text{P},d)^{\Delta}$的类似于算法 S 的算法。试讨论这些算法和习题 12 和 17 之间的关系。

▶19. [*27*] 试设计用于在(a)右穿线的二叉树中；(b)一个完全穿线的二叉树中，计算 P# 的类似于算法 S 的一个算法。如果可能，当 P 是树的一个随机节点时，你的算法的平均运行时间应当至多是一个小的常数。

20. [*23*] 修改程序 T 以便将栈保存在链接表中，而不是保存在连续的内存单元中。

▶21. [*33*] 试设计一个不使用任何辅助栈以中根序遍历一个无穿线二叉树的算法。以任何方式在遍历期间改变树节点的 LLINK 和 RLINK 字段都是允许的，而仅受以下条件的支配，即在你的算法遍历树之前和之后二叉树应当有在(2)中图示的传统表示。在树节点中没有其它的二进位可用作临时存储。

22. [*25*] 编写在习题 21 中给出算法的一个 MIX 程序并且和程序 S 和 T 比较它的执行时间。

23. [*22*] 试设计用于在一个右穿线的二叉树中插入到右边和插入到左边的类似于算法 I 的算法，假定诸节点有 LLINK，RLINK 和 RTAG 字段。

24. [*M20*] 如果以对称序而不是先根序给出节点 T 和 T'，定理 A 还正确吗？

25. [*M24*] 设$\mathscr{T}$是二叉树的集合，其中每个 info 字段属于一个给定的集合 S，而 S 是对关系“$\preceq$”线性有序的(请见习题 2.2.3-14)。给定$\mathscr{T}$中的任何树 T,T'。让我们现在定义 $T\preceq T'$当且仅当

i) T 为空；或

ii) T 和 T'非空，而且 $\text{info}(\text{root}(T)) \prec \text{info}(\text{root}(T'))$；或

iii) T 和 T'非空，$\text{info}(\text{root}(T))=\text{info}(\text{root}(T'))$，$\text{left}(T)\preceq\text{left}(T')$，而且 $\text{left}(T)$不等价于 $\text{left}(T')$；或

iv) T 和 T'非空，$\text{info}(\text{root}(T))=\text{info}(\text{root}(T'))$，$\text{left}(T)$等价于 $\text{left}(T')$，而且 $\text{right}(T)\preceq\text{right}(T')$。

这里 $\text{left}(T)$和 $\text{right}(T)$表示 T 的左子树和右子树。证明(a) $T\preceq T'$和 $T'\preceq T''$蕴涵 $T\preceq T''$；(b)T 等价于 T'当且仅当 $T\preceq T'$和 $T'\preceq T$；(c) 对于$\mathscr{T}$中的任何 T,T'，我们有 $T\preceq T'$或 $T'\preceq T$。[因此，如果在$\mathscr{T}$中的等价树被认为是相等，则关系$\preceq$导致在$\mathscr{T}$上的一个线性顺序。这个顺序有许多应用(例如，在代数表达式的简化方面)。当 S 有惟一元素，使得每个节点的“info”是相同的，我们就有等价性和相似性相同的特别情况。]

26. [*M24*] 考虑在上题中定义的顺序 $T\preceq T'$，证明类似于定理 A 的一个定理，请给出$T\preceq T'$的充分必要条件，并利用在习题 18 中定义的双重序。

▶27. [*28*] 试设计一个算法，检验两个给定的树 T 和 T'，借助于在习题 25 中定义的关系，试

确定 $T<T'$，$T>T'$ 还是 T 等价于 T'，假定两棵二叉树都是右穿线的。假定每个节点有 LLINK，RLINK，RTAG 和 INFO 字段；不使用辅助栈。

28. [*00*] 在算法 C 被用来复制一棵树之后，新的二叉树是否等价于原来的，或者相似于它？

29. [*M25*] 尽可能严格地证明，算法 C 是正确的。

▶30. [*22*] 试设计对无穿线树进行穿线的算法；例如它应当把(2)转换成(10)。注：当可能时，总使用像 P* 和 P$ 这样的记号，而不是重复像算法 T 那样的遍历算法的步骤。

31. [*23*] 试设计一个算法，它"擦除"一个右穿线二叉树。你的算法应当返回所有树节点，除对于 AVAIL 表的表头外，而且使表头标记一个空的二叉树。假定每个节点有 LLINK，RLINK 和 RTAG 字段；不要使用辅助栈。

32. [*21*] 假设一个二叉树的每个节点有四个链接字段：其中 LLINK 和 RLINK，如同在无穿线树中一样，指向左和右子树或 Λ；而 SUC 和 PRED，以对称序指向后继和前驱。(因此 SUC(P) = P$ 且 PRED(P) = $P。这样一棵树就比一棵穿线树包含更多信息。)试设计用于插入到这样一棵树的类似于算法 I 的算法。

▶33. [*30*] 有一种以上的方法来对一棵树穿线！使用在每个节点中的三个字段 LTAG，LLINK 和 RLINK，考虑下列表示：

LTAG(P)：和在穿线二叉树中一样地定义；

LLINK(P)：总是等于 P*；

RLINK(P)：和在无穿线二叉树中一样地定义。

试讨论对于这样一个表示的插入算法，并且，对于这个表示详细写出复制算法，即算法 C。

34. [*22*] 设 P 指向某个二叉树中的一个节点，且设 HEAD 指向一个空二叉树的表头。试给出一个算法，它(i)从 NODE(P)所在的无论什么样的树中删去 NODE(P)及其所有子树，而后(ii)把 NODE(P)和它的子树附加到 NODE(HEAD)上。假定问题中的所有二叉树都是右穿线的，而且在每个节点中有 LLINK，RTAG，RLINK 字段。

35. [*40*] 以类似于我们对二叉树的定义方式，定义一个*三叉树*(而且更一般地，对于任何 $t\geqslant 2$，*t 叉树*)，而且考察在本小节中所讨论的那些课题(包括在以上的习题中可找到的课题在内)，使得它们能以一种有意义的方式推广到 t 叉树来。

36. [*M23*] 习题 1.2.1-15 表明，字典序把一个集合 S 的良序推广到 S 的元素的 n 元组的良序上。上面的习题 25 说明使用相似的定义，在树节点中的信息的一个线性序，可以被推广到树的一个线性序。如果关系 $\prec$ 使 S 成为良序的，习题 25 的推广的关系是否使 $\mathscr{T}$ 成为良序的？

▶37. [*24*] (D. Ferguson) 如果需要两个计算机字来包含两个链接字段和一个 INFO 字段，对于有 n 个节点的一棵树表示(2)需要 $2n$ 个内存字。试设计使用较少空间的二叉树的表示方案，假定一个链接和一个 INFO 字段可装入到一个计算机字内。

2.3.2　树的二叉树表示

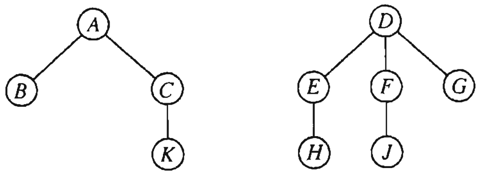
我们现在从二叉树转到普通的树。如同我们对它们所给出的定义那样，先回忆一下树和二叉树之间的基本差别：

1) 树总有一个根节点，所以它绝不会为空；树的每个节点可以有 0，1，2，3，…个儿子。

2) 二叉树可以为空,而且它的每个节点可以有0,1,2个儿子,我们区分"左"儿子和"右"儿子。

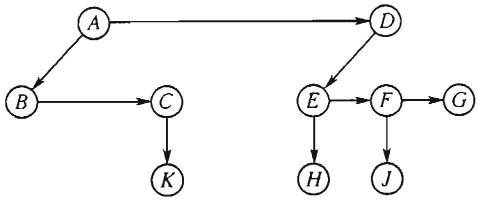
还回想一下,森林是零个或多个树的有序集合,在树的任何节点紧下面的子树形成森林。

如同一棵二叉树一样,存在表示任何森林的自然的方式。考虑有两棵树的以下的森林:



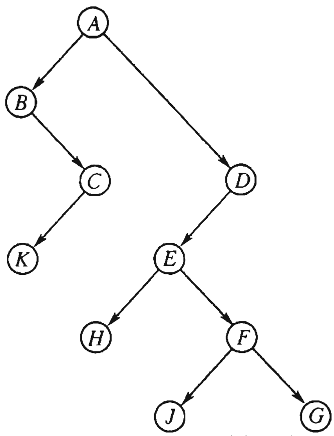
(1)

通过把每个家庭的孩子链接在一起和删除除了从父亲到头一个孩子的链接之外的垂直链接,得到对应的二叉树



(2)

然后,顺时针把这个图倾斜45°并且稍微地加以扭动,就得到二叉树



(3)

反过来,容易看到,通过逆转这个过程,任何二叉树对应于树的惟一的森林。

从(1)到(3)转换是极端重要的,它被称做森林和二叉树之间自然的对应;特别是它给出了树和特殊的二叉树类之间的对应,这就是有一个根但没有右子树的二叉树。(我们也可以稍微地改变观点,而令树的根对应于二叉树的表头,因此得到具有 $n+1$ 个节点的树和具有 n 个节点的二叉树之间的一一对应。)

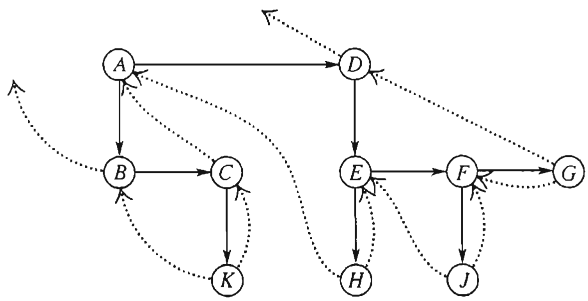
令 $F=(T_1,T_2,\cdots,T_n)$ 是树的森林。对应于 F 的二叉树 $B(F)$ 可严格地定义如下:

a)如果 $n=0$,则 $B(F)$为空。

b)如果 $n>0$,则 $B(F)$的根是 $\mathrm{root}(T_1)$;$B(F)$的左子树是 $B(T_{11},T_{12},\cdots,T_{1m})$,其中 $T_{11},T_{12},\cdots,T_{1m}$是 $\mathrm{root}(T_1)$的子树;且 $B(F)$的右子树是 $B(T_2,\cdots,T_n)$。

这些规则精确地描述了从(1)到(3)的变换。

无需作 45°的转动,而像在(2)中那样画二叉树图,有时是方便的。对应于(1)的穿线二叉树是(和图 24 作比较,对后者给予在方向上 45°的改变)



(4)

注意右穿线链接从*一个家庭的最右儿子通向父亲*。由于在左和右之间缺乏对称性,左穿线链接就没有这样自然的表示。

借助于森林(以及因此也包括树在内),在上一小节中被考虑的有关遍历的思想可以重新叙述。由于没有明显的位置来把根插入到它的后裔当中,因此对于中根序来说,没有简单的类似物;但是先根序和后根序可以以明显的方式进行。给出任意的非空森林,遍历它的两个基本方法可以定义如下:

先根序遍历	*后根序遍历*
访问第一棵树的根	遍历第一棵树的子树
遍历第一棵树的子树	访问第一棵树的根
遍历剩余的树	遍历剩余的树

为了理解这两个遍历方法的重要性,考虑通过嵌套的括弧来表达树结构的以下记号:

$$(A(B,C(K)),D(E(H),F(J),G)) \tag{5}$$

这个记号对应于森林(1):通过把信息写在它的根中,后面接上它的子树的表示,就可表示一棵树;一个非空森林的表示,是它的诸树表示的带括号的表,各个树之间以逗号分开。

如果以先根序遍历(1),我们以序列 $A\ B\ C\ K\ D\ E\ H\ F\ J\ G$ 来访问诸节点。这只不过是删除了括号和逗号的(5)。先根序是列出树节点的自然的方法:我们首先列出根,然后列出诸后裔。如果像在图 20(c)中那样通过缩进来表示树结构,则诸行就以先根序出现,本书的章节号本身(请见图 22)就是以先根序出现的。例如,2.3 小节后面跟着的是 2.3.1 小节,然后是 2.3.2,2.3.3,2.3.4,2.3.4.1,…,2.3.4.6,2.3.5,2.4 等等。

指出这样一点是有趣的,即先根序是历史悠久的概念,它有意义地被称为朝代顺序。在国王,公爵或伯爵死后,爵位传给第一个儿子,然后是此儿子的后裔。最后如果这些人全死了,就以同样的方式传给这个家庭中的其他儿子(英国的习惯也包括以和儿子相同的基础传给家庭中的女儿,除非女儿按顺序排在所有儿子的后边)。理论上说,我们可以取所有贵族的世系图,并且以先根序写出诸节点。然后如果只考虑现在还活着的人,我们就可得到御座继位的顺序(除了由退位法令所修改的之外)。

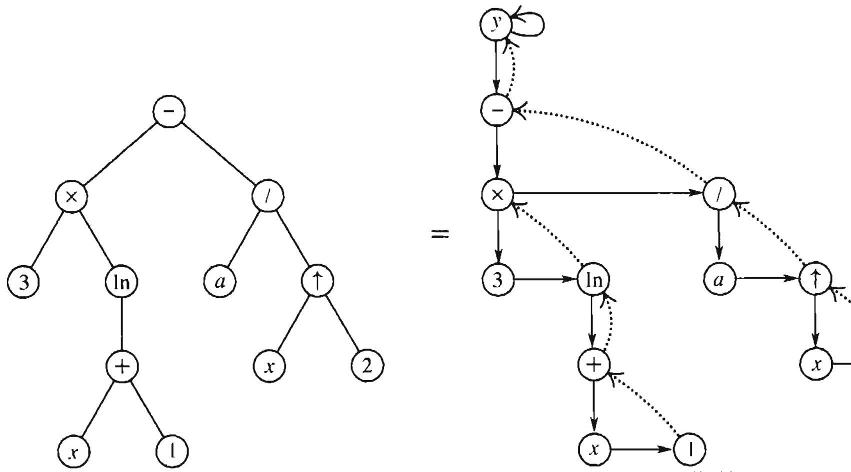
(1)中节点的后根序为 $B\,K\,C\,A\,H\,E\,J\,F\,G\,D$;这和先根序是类似的,只不过它对应于相似的括号表示

$$((B,(K)C)A,((H)E,(J)F,G)D) \tag{6}$$

其中一个节点恰出现于它的后裔后边而不是在它的前边。

先根序和后根序的定义非常好地同树和二叉树间的自然对应相配合,因为第一棵树的子树对应于左二叉子树,而且剩余的树对应于右二叉子树。通过把这些定义同2.3.1小节开始的定义作比较,我们发现以先根序遍历森林和以先根序遍历相应的二叉树完全相同,以后根序遍历森林和以中根序遍历相应的二叉树完全相同。在2.3.1小节中给出的诸算法因此可以不加修改径直使用。(注意对于树的后根序对应于二叉树的中根序,而非后根序。这是幸运的,因为我们已经知道,以后根序遍历二叉树是比较难的。)由于这个等价性,我们使用记号 P$ 来代表在树或森林中节点 P 的后根序后继,而它在二叉树中表示中根序的后继。

作为这些方法应用于实际问题的一个例子,我们将考虑代数公式的操作。把这样的公式当成树结构的表示,而不看成符号的一维或二维的配置,也不看成二叉树,是最适当的。例如:公式 $y=3\ln(x+1)-a/x^2$ 有树表示



(7)

这里左边的图是像图21那样的传统的树的图示,其中的二元操作符 +,−,×,/以及↑(后者表示乘幂)有对应于它们的操作数的两棵子树;一元操作符“ln”有一棵子树;变量和常量是终节点。右边的图示出等价的右穿线二叉树,包括附加的节点 y,它是这棵树

的表头。这个表头具有在 2.3.1-(8) 中所描述的形式。

重要的是要注意，尽管(7)中左边的树表面上类似于二叉树，但在这里我们把它作为树处理，并且通过(7)中右边的树所示的十分不同的二叉树来表示它。尽管我们可以直接基于二叉树的结构来编出代数操作的程序——即所谓的代数公式的“三地址代码”表示——但如果我们像在(7)中那样，使用代数公式的一般树的表示，在实践上就出现若干简化，因为在一棵树中后根序遍历是较为容易的。

(7)中左边树的节点是

$$\text{在先根序下}\quad - \quad \times \quad 3 \quad \ln \quad + \quad x \quad 1 \quad / \quad a \quad \uparrow \quad x \quad 2 \qquad (8)$$

$$\text{在后根序下}\quad 3 \quad x \quad 1 \quad + \quad \ln \quad \times \quad a \quad x \quad 2 \quad \uparrow \quad / \quad - \qquad (9)$$

像(8)和(9)这样的代数表达式是非常重要的，而且它们被称为“波兰记号”，因为形式(8)是由波兰逻辑学家 Jan Lukasiewicz 引进的。表达式(8)是公式(7)的*前缀记号*，而(9)是相应的*后缀记号*。在以后数章中我们将转回到波兰记号的有趣课题。至于现在，让我们只满足于知道，波兰记号是直接同树遍历的基本顺序有关的。

我们将假定，现在正在讨论的代数公式的树结构在 MIX 程序中有下列形式的节点

RTAG	RLINK	TYPE	LLINK
INFO			

(10)

这里的 RLINK 和 LLINK 具有通常的意义，而 RTAG 对于穿线链接是负的(对应于在算法语句中的 RTAG = 1)。TYPE 字段用来区分不同的节点类型：TYPE = 0 意味着这个节点表示常数，而 INFO 是该常数的值。TYPE = 1 意味着这个节点表示变量，而 INFO 是这个变量的五个字母的字符名。TYPE≥2 意味着这个节点表示操作符；INFO 是该操作符的字符名，而且 TYPE = 2,3,4,…用来区分不同的操作符 +，-，×，/等等。在这里我们先不关心在计算机的内存中树结构是如何建立的，因为在第 10 章中会对这个课题进行详尽分析；让我们只是假定在计算机内存中已经有树出现，而把输入和输出的问题推迟到稍后再谈。

我们现在将讨论代数操作的经典的例子，即寻求公式关于变量 x 的导数。用于代数微分的程序是为计算机写成的最初的符号操作程序当中的一个；这些程序早在 1952 年就已被使用了。微分的过程说明了代数操作的许多技术，而且在科学应用中它们也有重大的实用价值。

不熟悉微积分的读者可以把这个问题当成在公式操作中的一个抽象的习题，这个操作由下列规则来定义：

$$D(x) = 1 \qquad (11)$$

$$D(a) = 0, \quad \text{如果 } a \text{ 是常数或} \neq x \text{ 的变量} \qquad (12)$$

$$D(\ln u) = D(u)/u, \quad \text{如果 } u \text{ 是任意的公式} \qquad (13)$$

$$D(-u) = -D(u) \qquad (14)$$

$$D(u+v) = D(u) + D(v) \qquad (15)$$

$$D(u-v) = D(u) - D(v) \qquad (16)$$

$$D(u \times v) = D(u) \times v + D(v) \times u \tag{17}$$

$$D(u/v) \ = D(u)/v - (u \times D(v))/(v \uparrow 2) \tag{18}$$

$$D(u \uparrow v) = D(u) \times (v \times (u \uparrow (v-1))) + ((\ln u) \times D(v)) \times (u \uparrow v) \tag{19}$$

这些公式使我们能对由列出的操作符组成的任何公式 y 来计算导数 $D(y)$。在规则(14)中的“-”号是单目操作符,它不同于(16)中的二元的“-”;以下在树节点中我们将使用“neg”来代表单目的负。

不幸的是规则(11)~(19)并未告诉我们全部的事情。如果我们盲目地将这些公式应用于像

$$y = 3\ln(x+1) - a/x^2$$

这样的较为简单的公式,就会得到

$$D(y) = 0 \cdot \ln(x+1) + 3((1+0)/(x+1)) - \\ (0/x^2 - a(1(2x^{2-1}) + ((\ln x) \cdot 0)x^2))/(x^2)^2) \tag{20}$$

它是正确的但全然不令人满意。为了避免答案中这样多余的操作,我们必须认识加或乘零,乘 1,或者一次方这些特殊情况。这些简化把(20)归结为

$$D(y) = 3(1/(x+1)) - ((-(a(2x)))/(x^2)^2) \tag{21}$$

这是更可接受的但仍然不理想。真正令人满意的答案的概念是未明确地定义的,因为不同的数学家将喜欢以不同的方式来表达公式;但是,显然,(21)仍然有简化的余地。为了对公式(21)进行实质性的改进,需要编写代数简化程序(见习题 17),它将把(21)简化为例如

$$D(y) = 3(x+1)^{-1} + 2ax^{-3} \tag{22}$$

我们将满足于能产生出(21),而不是(22)的程序。

和通常一样,对这个算法的主要兴趣在于在计算机内执行这一过程的细节。在大多数计算机装置中都有许多高级语言和专用程序可以利用,同时还有内建的工具以简化像上面式子这样的代数操作;但是现在这个例子的目的是获得在基本的树操作方面的更多的经验。

以下算法的思想背景,是以后根序遍历树,在执行过程中形成每个节点的导数,直到最终计算出整个导数。使用后根序意味着在对操作数求微分之后我们将到达操作符节点(如“+”)。规则(11)到(19)意味着或早或迟,原来的公式的每个子公式都将被微分,所以也可以以后根序来求微分。

通过使用右穿线树,在算法的执行期间就可避免对栈的需要。另一方面,穿线树表示也有缺点,即我们将需要制作子树的副本;例如,在对于 $D(u \uparrow v)$的规则中,我们可能需要对 u 和 v 各复制三次。如果我们选定使用如在 2.3.5 小节中给出的列表表示,而不是树,就可能避免这样的复制。

算法 D(*微分法*) 如果 Y 是指向像上面描述的那样表示的公式的表头地址,而且 DY 是一个空树的表头的地址,则这个算法使 NODE(DY)指向表示 Y 关于变量“X”的解析

导数的树。

D1. [初始化] 置 P←Y$(即在后根序下树的第一个节点,它是在中根序下对应的二叉树的第一个节点)。

D2. [微分] 置 P1←LLINK(P);而且如果 P1≠Λ,也置 Q1←RLINK(P1)。然后执行以下描述的程序 DIFF[TYPE(P)]。(程序 DIFF[0],DIFF[1]等等将形成树关于根 P 的导数,而且将把指针变量 Q 置成导数的根的地址。首先设置变量 P1 和 Q1,以便简化 DIFF 程序的描述。)

D3. [恢复链接] 如果 TYPE(P)表示二元操作符,则置 RLINK(P1)←P2。(解释请见下一步。)

D4. [前进到 P$] 置 P2←P,P←P$。现在如果 RTAG(P2)=0(即,如果NODE(P2)在右边有一个兄弟),则置 RLINK(P2)←Q。(这是这一算法的巧妙部分:我们暂时地破坏树 Y 的结构,使得对于 P2 的导数的链接被保存以供将来使用。不见了的链接将在稍后的步骤 D3 中加以恢复。关于这个技巧的进一步讨论,请见习题 21。)

D5. [完成了吗?] 若 P≠Y,则返回步骤 D2。否则置 LLINK(DY)←Q 和 RLINK(Q)←DY,RTAG(Q)←1。 ▌

在算法 D 中所描述的过程仅仅是在步骤 D2 中调用的处理程序 DIFF[0],DIFF[1],…所实施的微分操作的基础程序。在许多方面,算法 D 就像是在 1.4.3 小节所讨论的那样,是一个解释系统或机器模拟程序的控制程序,但它遍历树而不是执行简单的指令序列。

为了完成算法 D,我们必须定义真正求微分的程序。在以下的讨论中,语句“P 指向一棵树”指的是,NODE(P)是作为右穿线二叉树被存储的一棵树的根,尽管迄今为止 RLINK(P)和 RTAG(P)就这棵树而言还是无意义的。我们将使用一个*树构造函数*,此函数通过把较小的一些树连接在一起形成新的树:命 x 表示某种类型的节点,或者是常数、变量,或者是操作符,并令 U 和 V 表示树的指针。接着,

TREE(x,U,V)以 x 作为它的根节点且 U 和 V 作为根的子树,形成一棵新的树:

W⇐AVAIL,INFO(W)←x,LLINK(W)←U,RLINK(U)←V,RTAG(U)←0,
RLINK(V)←W,RTAG(V)←1。

TREE(x,U)相似地但仅以一个子树形成新的树:

W⇐AVAIL,INFO(W)←x,LLINK(W)←U,RLINK(U)←W,RTAG(U)←1。

TREE(x)以 x 作为终根节点来形成新的树:

W⇐AVAIL,INFO(W)←x,LLINK(W)←Λ。

而且,依赖于 x,适当地设置 TYPE(W)。在所有情况下,TREE 的值是 W,即指向刚刚构造的树的指针。读者应当仔细地研究这三个定义,因为它们说明树的二叉树表示。另一个函数 COPY(U)生成由 U 所指向的树的一个副本,并且把由此建立的对于这棵树的一个指针作为它的值。基本的函数 TREE 和 COPY 使得逐步地构造用于求公式的导数的树变得容易了。

零操作符(常数和变量) 对于这些操作,NODE(P)是一个终节点,而且在操作之前 P1,P2,Q1 和 Q 的值是不相干的。

DIFF[0]:(NODE(P)是常数。)置 Q←TREE(0)。

DIFF[1]:(NODE(P)是变量。)如果 INFO(P)="X",则置 Q←TREE(1),否则置 Q←TREE(0)。

单目操作符(对数和取负) 对于这些操作,NODE(P)有一个由 P1 指向的儿子 U,且 Q 指向 $D(U)$。在操作之前 P2 和 Q1 的值是不相干的。

DIFF[2]:(NODE(P)是"ln"。)若 INFO(Q)≠0,置 Q←TREE("/",Q,COPY(P1))。

DIFF[3]:(NODE(P)是"neg"。)如果 INFO(Q)≠0,则置 Q←TREE("neg",Q)。

二元操作符(加法,减法,乘法,除法,乘幂) 对于这些操作,NODE(P)有两个儿子 U 和 V,分别由 P1 和 P2 来指向;Q1 和 Q 分别指向 $D(U)$,$D(V)$。

DIFF[4]:("+"操作。) 如果 INFO(Q1)=0,则置 AVAIL⇐Q1。否则若 INFO(Q)=0,则置 AVAIL⇐Q 及 Q←Q1;否则置 Q←TREE("+",Q1,Q)。

DIFF[5]:("−"操作。) 如果 INFO(Q)=0,置 AVAIL⇐Q,以及 Q←Q1。否则,如果 INFO(Q1)=0,则置 AVAIL⇐Q1 和置 Q←TREE("neg",Q);否则置 Q←TREE("−",Q1,Q)。

DIFF[6]:("×"运算。) 如果 INFO(Q1)≠0,则置 Q1←MULT(Q1,COPY(P2))。然后如果 INFO(Q)≠0,则置 Q←MULT(COPY(P1),Q)。然后转到 DIFF[4]。

这里 MULT(U,V)是一个构造 U×V 的树的新函数,但也进行测试看看是否 U 或 V 等于 1:

如果 INFO(U)=1 且 TYPE(U)=0,则置 AVAIL⇐U 和 MULT(U,V)←V;

如果INFO(V)=1 且 TYPE(V)=0,则置 AVAIL⇐V 和 MULT(U,V)←U;否则置

MULT(U,V)←TREE("×",U,V)

DIFF[7]:("/"操作。) 如果 INFO(Q1)≠0,则置 Q1←TREE("/",Q1,COPY(P2))。然后如果 INFO(Q)≠0,则置

Q←TREE("/",MULT(COPY(P1),Q),TREE("↑",COPY(P2),TREE(2)))

然后转到 DIFF(5)。

DIFF[8]:("↑"操作。)请见习题 12。

我们通过展示所有上面的操作如何容易地变换成计算机程序,来结束本小节,并且仅以 MIX 机器语言为基础,从头开始。

程序 D(求微分) 下列 MIXAL 程序实现算法 D,且有 rI2≡P,rI3≡P2,rI4≡P1,rI5≡Q,rI6≡Q1。为方便起见,对计算的顺序已稍作调整。

```
* DIFFRENTIATION IN A RIGHT-THREADED TREE
LLINK    EQU    4:5        字段的定义,见(10)
RLINK    EQU    1:2
RLINKT   EQU    0:2
TYPE     EQU    3:3
```

```
006  * MAIN   CONTROL  ROUTINE        D1. 初始化
007    D1     STJ      9F             把整个过程当做子程序处理
008           LD4      Y(LLINK)       P1←LLINK(Y),准备求 Y$
009    1H     ENT2     0,4            P←P1
010    2H     LD4      0,2(LLINK)     P1←LLINK(P)
011           J4NZ     1B             如果 P1≠Λ,则重复
012    D2     LD1      0,2(TYPE)      D2. 求微分
013           JMP      *+1,1          转到 DIFF[TYPE(P)]
014           JMP      CONSTANT       转到对于 DIFF[0]的表入口
015           JMP      VARIABLE       转到对于 DIFF[1]的表入口
016           JMP      LN             转到对于 DIFF[2]的表入口
017           JMP      NEG            转到对于 DIFF[3]的表入口
018           JMP      ADD            转到对于 DIFF[4]的表入口
019           JMP      SUB            转到对于 DIFF[5]的表入口
020           JMP      MUL            转到对于 DIFF[6]的表入口
021           JMP      DIV            转到对于 DIFF[7]的表入口
022           JMP      PWR            转到对于 DIFF[8]的表入口
023    D3     ST3      0,4(RLINK)     D3. 恢复链接。RLINK(P1)←P2
024    D4     ENT3     0,2            D4. 前进到 P$。P2←P
025           LD2      0,2(RLINKT)    P←RLINKT(P)
026           J2N      1F             如果 RTAG(P) = 1 则转移
027           ST5      0,3(RLINK)       否则置 RLINK(P2)←Q
028           JMP      2B             注意 NODE(P$ )将是终节点
029    1H     ENN2     0,2
030    D5     ENT1     -Y,2           D5. 完成了吗?
031           LD4      0,2(LLINK)     P1←LLINK(P),为步骤 D2 做准备
032           LD6      0,4(RLINK)     Q1←RLINK(P1)
033           J1NZ     D2             如果 P≠Y 转到 D2;
034           ST5      DY(LLINK)        否则置 LLINK(DY)←Q
035           ENNA     DY
036           STA      0,5(RLINKT)    RLINK(Q)←DY,RTAG(Q)←1
037    9H     JMP      *              从求微分子程序离开 ▌
```

这个程序的下一个部分包含基本的子程序 TREE 和 COPY。按照被构造的树的子树的个数,前者有三个入口 TREE0,TREE1 和 TREE2。不论使用子程序的哪一个入口,rA 都将包含一个特殊常数的地址,它指出何种类型的节点形成正在构造的树的根;这些特殊常数出现于行 105 ~ 124 中。

```
038  * BASIC  SUBROUTINES FOR TREE CONSTRUCTION
```

```
TREE0   STJ    9F            TREE(rA)函数:
        JMP    2F
TREE1   ST1    3F(0:2)       TREE(rA,rI1)函数:
        JSJ    1F
TREE2   STX    3F(0:2)       TREE(rA,rX,rI1)函数:
3H      ST1    *(RLINKT)     RLINK(rX)←rI1,RTAG(rX)←0
1H      STJ    9F
        LDXN   AVAIL
        JXZ    OVERFLOW
        STX    0,1(RLINKT)   RLINK(rI1)←AVAIL,RTAG(rI1)←1
        LDX    3B(0:2)
        STA    *+1(0:2)
        STX    *(LLINK)      置下一个根节点的 LLINK
2H      LD1    AVAIL         rI1⇐AVAIL
        J1Z    OVERFLOW
        LDX    0,1(LLINK)
        STX    AVAIL
        STA    *+1(0:2)      把根的 info 复制到新节点中
        MOVE   *(2)
        DEC1   2             恢复 rI1 以指向新节点
9H      JMP    *             从 TREE 离开,rI1 指向新树
COPY1   ENT1   0,4           COPY(P1),COPY 的特殊入口
        JSJ    COPY
COPY2   ENT1   0,3           COPY(P2),COPY 的特殊入口
COPY    STJ    9F            COPY(rI1)函数:
 ⋮       ⋮                   (见习题 13)
9H      JMP    *             从 COPY 离开,rI1 指向新树
CON0    CON    0             表示常数“0”的节点
        CON    0
CON1    CON    0             表示常数“1”的节点
        CON    1
CON2    CON    0             表示常数“2”的节点
        CON    2
LOG     CON    2(TYPE)       表示“ln”的节点
        ALF    LN
NEGOP   CON    3(TYPE)       表示“neg”的节点
        ALF    NEG
PLUS    CON    4(TYPE)       表示“+”的节点
```

```
          ALF      +
MINUS     CON      5(TYPE)      表示“-”的节点
          ALF      -
TIMES     CON      6(TYPE)      表示“×”的节点
          ALF      *
SLASH     CON      7(TYPE)      表示“/”的节点
          ALF      /
UPARROW   CON      8(TYPE)      表示“↑”的节点
          ALF      **
```

程序的剩余部分对应于求微分的程序 DIFF[0],DIFF[1],…;这些程序被写成在处理二元操作符之后就把控制转回到步骤 D3,否则返回步骤 D4。

```
 * DIFFERENTIATION ROUTINES
VARIABLE  LDX      1,2
          ENTA     CON1
          CMPX     2F           INFO(P) = “X”吗?
          JE       *+2          如果是,则调用 TREE(1)
CONSTANT  ENTA     CON0         调用 TREE(0)
          JMP      TREE0
1H        ENT5     0,1          Q←新树的单元
          JMP      D4           回到控制程序
2H        ALF         X
LN        LDA      1,5
          JAZ      D4           如果 INFO(Q) = 0 转回控制程序
          JMP      COPY1          否则置 rI1←COPY(P1)
          ENTX     0,5
          ENTA     SLASH
          JMP      TREE2        rI1←TREE(“/”,Q,rI1)
          JMP      1B           Q←rI1,返回到控制程序
NEG       LDA      1,5
          JAZ      D4           如果 INFO(Q) = 0,则返回
          ENTA     NEGOP
          ENT1     0,5
          JMP      TREE1        rI1←TREE(“neg”,Q)
          JMP      1B           Q←rI1,返回到控制程序
ADD       LDA      1,6
          JANZ     1F           除非 INFO(Q1) = 0 否则转移
3H        LDA      AVAIL        AVAIL⇐Q1
```

```
        STA     0,6(LLINK)
        ST6     AVAIL
        JMP     D3              返回到控制程序,二元操作符
1H      LDA     1,5
        JANZ    1F              除非 INFO(Q) = 0 否则转移
2H      LDA     AVAIL           AVAIL⇐Q
        STA     0,5(LLINK)
        ST5     AVAIL
        ENT5    0,6             Q←Q1
        JMP     D3              返回到控制程序
1H      ENTA    PLUS            准备调用 TREE("+",Q1,Q)
4H      ENTX    0,6
        ENT1    0,5
        JMP     TREE2
        ENT5    0,1             Q←TREE("±",Q1,Q)
        JMP     D3              返回到控制程序
SUB     LDA     1,5
        JAZ     2B              如果 INFO(Q) = 0 则转移
        LDA     1,6
        JANZ    1F              除非 INFO(Q) = 0,否则转移
        ENTA    NEGOP
        ENT1    0,5
        JMP     TREE1
        ENT5    0,1             Q←TREE("neg",Q)
        JMP     3B              AVAIL⇐Q1 并返回
1H      ENTA    MINUS           准备调用 TREE("-",Q1,Q)
        JMP     4B
MUL     LDA     1,6
        JAZ     1F              如果 INFO(Q) = 0 则转移;否则
        JMP     COPYP2            置 rI1←COPY(P2)
        ENTA    0,6
        JMP     MULT            rI1←MULT(Q1,COPY(P2))
        ENT6    0,1             Q1←rI1
1H      LDA     1,5
        JAZ     ADD             如果 INFO(Q) = 0 则转移;
        JMP     COPYP1            否则置 rI1←COPY(P1)
        ENTA    0,1
        ENT1    0,5
```

```
          JMP    MULT          rI1←MULT(COPY(P1),Q)
          ENT5   0,1           Q←rI1
          JMP    ADD
MULT      STJ    9F            MULT(rA,rI1)子程序:
          STA    1F(0:2)       令 rA≡U,rI1≡V
          ST2    8F(0:2)       保存 rI2
1H        ENT2   *             rI2←U
          LDA    1,2           测试是否 INFO(U) = 1
          DECA   1
          JANZ   1F
          LDA    0,2(TYPE)     以及是否 TYPE(U) = 0
          JAZ    2F
1H        LDA    1,1           如果不是,测试是否 INFO(V) = 1
          DECA   1
          JANZ   1F
          LDA    0,1(TYPE)     以及是否 TYPE(V) = 0
          JANZ   1F
          ST1    *+2(0:2)      如果是,交换 U↔V
          ENT1   0,2
          ENT2   *
2H        LDA    AVAIL         AVAIL⇐U
          STA    0,2(LLINK)
          ST2    AVAIL
          JMP    8F            结果为 V
1H        ENTA   TIMES
          ENTX   0,2
          JMP    TREE2         结果是 TREE("×",U,V)
8H        ENT2   *             恢复 rI2 的设定
9H        JMP    *             以结果在 rI1 中离开 MULT ▎
```

其它两个程序 DIV 和 PWR 是相似的,因而把它们留作习题(请见习题 15 和 16)。

习　题

▶ 1.[*20*] 正文给出 $B(F)$,即对应于一个森林 F 的二叉树的形式定义。试给出逆转这个过程的形式定义;换句话说,定义 $F(B)$,即对应于二叉树 B 的森林。

▶ 2.[*20*] 在 2.3 节我们定义了对于森林的杜威十进记号,在习题 2.3.1-5 定义了对于二叉树的同一记号。于是在(1)中的节点"J"被表示为"2.2.1",而在等价的二叉树(3)中它被表示为

"11010"。如果可能，试给出一个规则，直接地把树和二叉树之间的对应表达为两个杜威十进记号间的对应关系。

3. [*22*] 森林的节点的杜威十进记号和这些节点的先根序与后根序间的关系是什么？

4. [*19*] 下列命题是真还是假？"树的终节点在先根序下和在后根序下处于相同的相对位置。"

5. [*23*] 森林和二叉树之间的另一个对应可以通过令 RLINK(P)指向 NODE(P)的最右儿子，而令 LLINK(P)指向左边最靠近的兄弟来定义。令 F 是以这种方式对应于二叉树 B 的一个森林。对于 B 的节点，什么顺序对应于 F 的(a)先根序，(b)后根序？

6. [*25*] 令 T 是非空的二叉树，其中每个节点有 0 个或 2 个儿子。如果我们把 T 当作通常的树，它(通过自然的对应)对应于另一二叉树 T'。T(作为二叉树定义)的节点的先根序，中根序，后根序和对于 T'的节点的同样三种次序之间有任何简单的关系吗？

7. [*M20*] 如果我们说每个节点在树中都居于它的后裔之前，则一个森林可以认为是偏序的。当森林的节点是以(1)先根序，(b)后根序，(c)逆先根序，(d)逆后根序列出时，它们是否(如在 2.2.3 小节中所定义的那样)已被拓扑排序？

8. [*M20*] 习题 2.3.1-25 说明，一个二叉树的各个节点中所存的信息之间的一个顺序，如何被推广成为所有二叉树的线性顺序。在自然对应之下同样的构造导致所有树的顺序。借助于树，试重新阐述该习题的定义。

9. [*M21*] 试证明在森林中非终节点的总数同在对应的无穿线二叉树中右链接等于 Λ 的总数之间有简单的关系。

10. [*M23*] 令 F 是在先根序下其节点为 $u_1, u_2, \cdots, u_n$ 的树之森林，并令 F' 是在先根序下其节点为 $u'_1, u'_2, \cdots, u'_n$ 的森林。令 $d(u)$表示节点 u 的度(即儿子的个数)。借助于这些思想，阐述和证明与定理 2.3.1A 相似的一个定理。

11. [*15*] 对应于公式 $y = e^{-x^2}$ 画出类似于在(7)中所画的树。

12. [*M21*] 试给出对于程序 DIFF[8]("↑"操作)的说明，它在正文的算法中被省略了。

▶ 13. [*26*] 试编写 COPY 子程序的 MIX 程序(它可放入行 063 ~ 104 间的程序正文中)。[提示：以适当的初始条件，把算法 2.3.1C 修改成右穿线二叉树的情况。]

▶ 14. [*M21*] 为复制具有 n 个节点的树，习题 13 的程序要花费多长时间？

15. [*23*] 对应于正文中描述的 DIFF[7]，试编写 DIV 子程序的 MIX 程序。(这个程序应当被加到正文中的程序的行 217 之后。)

16. [*24*] 对应于习题 12 所描述的 DIFF[8]，试编写 PWR 程序的 MIX 程序。(这个程序应当在习题 15 的解之后加进正文的程序中。)

17. [*M40*] 试编写能把例如(20)或(21)简化为(22)的实现代数简化的程序。[提示：对于每个节点包括一个新字段，表示它的系数(对于求和项)或它的指数(对于在乘积中的因子)。应用代数恒等式，好比以 $v \ln u$ 来代替 $\ln(u \uparrow v)$；可能时通过使用等价的加法和乘法操作来删去 $-$，/，↑ 以及 neg 操作。把 + 和 × 变成 n 元的而不是二元操作符；通过把它们的操作数排成树的顺序而合并同类项(习题 8)；某些和与积现在将简化为 0 或 1，因而也许提供更进一步的简化。还有其它调整，比如以一个乘积的对数来代替对数之和，也可以进行。]

▶ 18. [*25*]对于 $1 \le j \le n$，由 n 个链接 PARENT[j]所确定的一棵有向树隐含定义了一棵有序树，如果在每个家庭中的节点按照它们的位置是有序的话。试设计一个有效的算法，构造包含在先根序下这个有序树的节点的双重链接的循环表。例如，给定

$$j = 1\quad 2\quad 3\quad 4\quad 5\quad 6\quad 7\quad 8$$
$$\text{PARENT}[j] = 3\quad 8\quad 4\quad 0\quad 4\quad 8\quad 3\quad 4$$

你的算法应当产生出

$$\text{LLINK}[j] = 3\quad 8\quad 4\quad 6\quad 7\quad 2\quad 1\quad 5$$
$$\text{RLINK}[j] = 7\quad 6\quad 1\quad 3\quad 8\quad 4\quad 5\quad 2$$

而且它还应当报告，根节点是 4。

19. [*M35*] 自由格是个数学系统，(对于本习题的目的来说)它可以简单地定义为由变量和两个抽象的二元操作符“∨”及“∧”组成的所有公式的集合。以下列规则定义在自由格中某些公式 X 和 Y 之间的关系“$X \ge Y$”：

i) $X\vee Y\ge W\wedge Z$ 当且仅当 $X\vee Y\ge W$ 或 $X\vee Y\ge Z$ 或者 $X\ge W\wedge Z$ 或 $Y\ge W\wedge Z$；

ii) $X\wedge Y\ge Z$ 当且仅当 $X\ge Z$ 且 $Y\ge Z$；

iii) $X\ge Y\vee Z$ 当且仅当 $X\ge Y$ 或 $X\ge Z$；

iv) $x\ge Y\wedge Z$ 当且仅当 $x\ge Y$ 或 $x\ge Z$，x 是变量；

v) $X\vee Y\ge z$ 当且仅当 $X\ge z$ 或 $Y\ge z$，z 是变量，；

vi) $x\ge y$ 当且仅当 $x=y$，x 和 y 都是变量。

例如，我们发现 $a\wedge(b\vee c)\ge(a\wedge b)\vee(a\wedge c)\not\ge a\wedge(b\vee c)$。

给定在自由格中的两个公式 X 和 Y，试设计测试是否 $X\ge Y$ 的算法。

▶ 20. [*M22*] 试证明如果 u 和 v 是一个森林的节点，u 是 v 的祖宗当且仅当在先根序下 u 居于 v 之前，而在后根序下，u 在 v 之后。

21. [*25*] 算法 D 控制二元操作符、一元操作符以及零元操作符的微分运算，因此也控制节点有 2，1 和 0 的度数的树的微分运算；但是没有明确指出三元操作符以及度数更高的节点如何来处理控制。(例如习题 17 提议把加法和乘法改成有任意个数的操作数的操作符。)是否有可能以简单的方式把算法 D 推广成使它可处理度数大于 2 的操作符？

▶ 22. [*M26*] 如果 T 和 T' 是树，而且如果存在一个从 T 的节点到 T' 的节点的一一对应函数 f，使得 f 保持先根序和后根序，则我们就说 T 可被嵌入 T' 中，并写成 $T\subseteq T'$。(换句话说，对于 T 来说先根序下 u 居于 v 之前当且仅当对于 T' 来说，在先根序下 $f(u)$ 居于 $f(v)$ 之前，而且对于后根序来说，同一事实成立，请见图 25。)

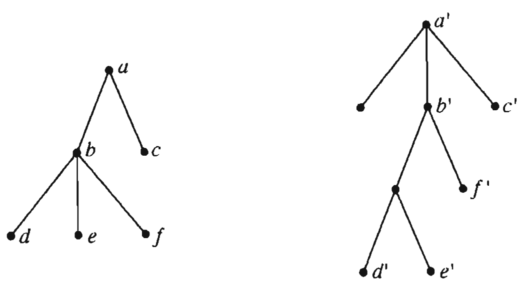


图 25　一棵树嵌入另一棵树(请见习题 22)

如果 T 有一个以上节点，令 $l(T)$ 是 root(T)最左子树，并令 $r(T)$ 是 T 的剩余部分，即去掉 $l(T)$ 之后的 T。试证明，T 可被嵌入 T' 中，如果(i) T 刚好只有一个节点，或(ii) T 和 T' 两者都有

一个以上的节点,且或者 $T\subseteq l(T')$,或者 $T\subseteq r(T')$,或者($l(T)\subseteq l(T')$且 $r(T)\subseteq r(T')$)。试问反之成立吗?

2.3.3 树的其它表示

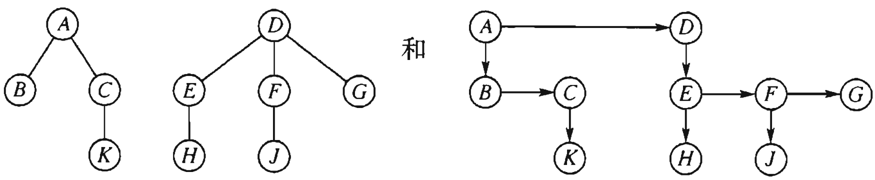
除了在上一小节给出的 LLINK-RLINK(左儿子-右兄弟)方法外,在计算机内表示树结构还有许多方法。和通常一样,对于表示的适当选择高度依赖于我们要对树实施什么类型的操作。在这一小节里,我们将考虑已被证明特别有用的树表示方法。

首先我们可以使用顺序存储术。像在线性表的情况那样,当我们需要这样一种树结构的紧凑表示——即它不会在程序执行期间由于大小和形状的改变而产生急剧的动态变化——时,则这种分配方式是最适当的。有许多情况,我们实质上需要树结构固定不变的表格,以便在程序内访问,因此在存储中这些树所要求的形式就依赖于对表格进行考察的方式。

树(以及森林)最普通的顺序表示实质上对应于 LLINK 字段的省略,并代之以使用连续的地址。例如,让我们再次查看在上一小节里考虑过的森林

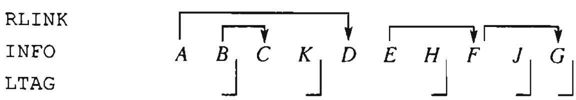
$$(A(B,C(K)),D(E(H),F(J),G)) \tag{1}$$

它有如下的树的图示:



(2)

先根序的顺序表示有以先根序出现的节点,而且在每个节点中有 INFO,RLINK 和 LTAG 字段:

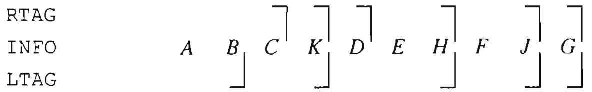


(3)

这里非空的 RLINK 已由箭头表示,而且 LTAG = 1(对于终节点)由"┘"指出。LLINK 是没有必要的,因为它或者为空,或者指向顺序中的下一项。对(1)和(3)作一比较是有教益的。

这种表示有若干个有趣的性质。首先,一个节点的所有子树立即出现于该节点之后,使得在原来的森林内的所有子树在连续的块中出现。[试把这一点同(1)中和图 20(b)中"嵌套的括弧"作比较。]其次,注意到在(3)中的箭头绝不彼此交叉;这一般说来都是对的,因为在二叉树中在先根序下 X 和 RLINK(X)之间的所有节点都位于 X 的左子树上,因此没有往外的箭头将从树的这一部分冒出。第三,我们可能发现,LTAG 字段,表示节点是否终节点,是多余的,因为"┘"仅在森林的末尾以及居于向下指的箭头之前才出现。

其实,这些说明表明,RLINK 字段本身几乎也是多余的;为表示这个结构我们真正需要的是 RTAG 和 LTAG。因此有可能从少得多的数据来演绎出(3)来:

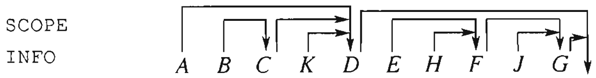


(4)

当我们从左到右地扫描(4)时,带有 RTAG≠“˥”的位置对应于必须被填入的非空的 RLINK。每次当我们扫描有 LTAG=“˩”的一项时,就应完成不完备的 RLINK 的最新的实例。(因此不完备的 RLINK 的位置可保存在栈上。)我们实际上再次证明了定理 2.3.1A。

除非顺序地扫描整个森林,否则,RLINK 或 LTAG 在(3)中是多余的事实对我们来说帮助极小甚至毫无用处,因为为推演出未出现的信息需要额外的计算。因此我们通常需要(3)中所有的数据。然而,明显地有某些浪费的空间,因为对于这个特定的森林来说,有多于一半的 RLINK 字段等于 Λ。利用浪费了的空间有两种常用的方法。

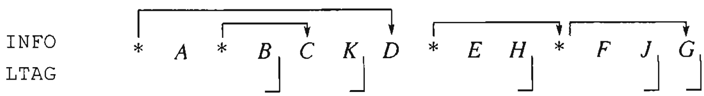
1)以在该节点之下在子树之后的地址来填入每个节点的 RLINK。这个字段现在通常称为“SCOPE”而不是 RLINK,因为它指出每个节点的“影响”(后裔)的右边界。代替(3),我们将有



(5)

箭头仍然彼此不交叉。而且 LTAG=“˩”通过条件 SCOPE(X)=X+c 来表征,其中 c 是每个节点的字数。使用这个 SCOPE 想法的例子出现于习题 2.4-12 中。

2)通过删去 RLINK 字段减少每个节点的大小,并且在原先有非空 RLINK 的节点紧前面添加特殊的“链接”节点:



(6)

这里“*”表示特殊的链接节点,其 INFO 将它们表征为箭头示出的链接。如果(3)的 INFO 和 RLINK 字段大约占据相同数量的空间,则改变成(6)的净效果是消耗较小的内存,因为“*”节点的数目总是少于非“*”节点的数目。表示(6)有点类似于 MIX 这样的单地址计算机中的指令序列,而“*”节点对应于条件转移指令。

类似于(3)的另一个顺序表示可以通过省略 RLINK 而不是 LLINK 来设计。在这种情况下,以新的顺序来列出森林的节点,这个顺序可以叫做*家庭序*,因为每个家庭成员出现在一起。对于任何森林的家庭序可以递归地定义如下:

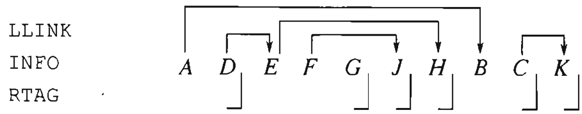
访问第一棵树的根。

(以家庭序)遍历剩余的树。

遍历(以家庭序)第一棵树的根的子树。

(试把这同在上一小节给出的先根序和后根序的定义作比较,家庭序和在对应的二叉树中的后根序的逆相同。)

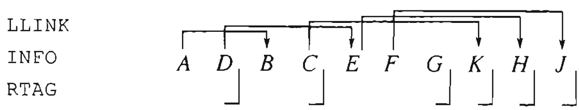
树(2)的*家庭序的顺序表示为*



(7)

在这种情况下 RTAG 的项用作界定家庭。家庭序通过列出森林中所有树的根开始,然后通过列出各个家庭来继续,逐次地选出最近出现的而其家庭尚未被列出的节点的家庭。由此得出,LLINK 箭头将绝不交叉;而且以相似的方式可带出先根序表示的其它性质。

代替使用家庭序,我们还可以简单地从左到右地列出节点,一次一层,这称为“层次序”[参见 G. Salton, *CACM* **5** (1962), 103 ~ 114],而且(2)的*层次序的顺序表示为*



(8)

这很像(7),但是家庭是以先进先出方式而不是后进先出方式选择的。(7)或(8)都可当做对于树的线性表的顺序表示的自然模拟。

读者将容易看出,如何设计遍历和分析如上面这样顺序地表示的树的算法,因为 LLINK 和 RLINK 信息实际上可以利用,就如同我们有充分地链接的树结构那样。

另一个顺序的方法,叫做*带度数的后根序*,它稍微不同于上面介绍的这些技术。我们以后根序列出节点并且给出每个节点的度以代替链接。

DEGREE	0	0	1	2	0	1	0	1	0	3
INFO	*B*	*K*	*C*	*A*	*H*	*E*	*J*	*F*	*G*	*D*

(9)

关于这足以表征树结构的证明,请见习题 2.3.2-10。如同在下列算法所述那样,这个顺序对于在树的节点上定义的函数的“由底向上”的计算是有用的。

算法 F(*计算在树中局部定义的函数*) 假设 f 是树的节点的函数,使得在树的节点 x 处 f 的值仅仅依赖于 x 和 x 的儿子的 f 值。下列算法,利用辅助栈,计算在非空的森林中每个节点处的 f。

F1. [初始化] 置栈为空,并令 P 指向在后根序下的森林的头一个节点。

F2. [计算 f] 置 $d \leftarrow$ DEGREE(P)。(第一次抵达这一步骤时,d 将为零。一般来说,当我们到达这一点时,下面的叙述总为真,即栈的顶部的 d 个元素,从栈顶往下是 $f(x_d), \cdots, f(x_1)$,其中 $x_1, \cdots, x_d$ 是从左到右 NODE(P)的儿子。)利用在栈上找到的 $f(x_d), \cdots, f(x_1)$ 的值,计算 f(NODE(P))。

F3. [更新栈] 从栈上删去顶部的 d 个项;然后把值 f(NODE(P))放入栈顶。

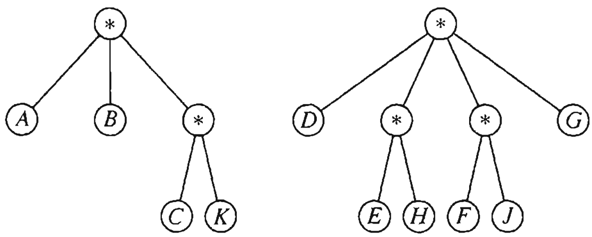
F4. [前进] 如果 P 是后根序下的最后节点,则终止这个算法。(从顶向下,栈中将包含 $f(\mathrm{root}(T_m)), \cdots, f(\mathrm{root}(T_1))$,其中 $T_1, \cdots, T_m$ 是给定的森林的诸树。)否则置

P 为在后根序下它的后继(在表示(9)中这只不过是 P←P + c),并返回到 F2。▎

算法 F 的正确性可由对被处理的树大小用归纳法得出(请见习题 16)。这个算法和上一小节的求微分过程(算法 2.3.2D)有令人惊异的相似性,它计算一个密切相关的函数类型;请见习题 3。和计算在后缀表示下的算术表达式相关联,相同的思想被用于许多解释程序中;在第 8 章中我们还将回到这个题目上来。也请参见习题 17,它给出和算法 F 相似的另一个重要的过程。

因此我们已经看到树和森林的各种顺序表示。也还有一些链接形式的表示,我们现在马上就要加以考虑。

第一个思想与把(3)变成(6)的变换有关:我们从所有的非终节点删去 INFO 字段,而把这个信息作为新的终节点放到以前节点的下面。例如,树(2)将变成



(10)

这个新的形式表明,我们可以假定(不失一般性),在一个树结构中所有 INFO 都出现于它的终节点中。因此,在 2.3.2 小节的自然的二叉树表示中,LLINK 和 INFO 字段是相互排斥的。因此在每个节点中它们可以共享同一字段。一个节点可以有下列字段:

LTAG	LLINK 或 INFO	RLINK

其中符号 LTAG 表明第二个字段是否是个链接。(例如把这个表示与 2.3.2 小节的(10)的两字格式相比较。)通过把每个 INFO 从 5 个字节(加 1 个符号)消减成 3 个字节,我们可以把每个节点放入一个字内。然而,注意,现在有 15 个节点而不是 10 个;森林(10)需要 15 个字的内存,而(2)需要 20 字的内存,而且与前者的 30 字节的 INFO 相比,后者使用 50 字节的 INFO。除非打算浪费多余的 INFO 空间,(10)中未获得在内存空间上的任何实际的收益;在(10)中被代替的 LLINK 的删除是以在增加的节点中几乎相同个数的新的 RLINK 的花销为代价的。在习题 4 中讨论了这两个表示之间的差别的精确细节。

在树的标准二叉树表示中,LLINK 字段可以更精确地称为 LCHILD 字段,因为它从父亲节点指向它最左的儿子。最左的儿子通常是在树中最年轻的儿子,因为比起在右边的儿子来,把节点插入家庭的左边是较为容易的;所以缩写 LCHILD 可以看做"最后的"或"最小的"儿子。

树结构的许多应用颇为经常地要求树中向上以及向下的访问。穿线树给了我们向上的能力,但是速度不很快;如果在每个节点中我们有第三个链接 PARENT,有时我们可以做得更好些。这就导致了*三重链接树*,其中每个节点有 LCHILD,RLINK 和 PARENT 链接。图 26 示出(2)的三重链接树表示。关于使用三重链接树的例子,请见 2.4 节。

显然,PARENT 链接本身就足以完全确定任何有向树(或森林)。因为如果我们知道

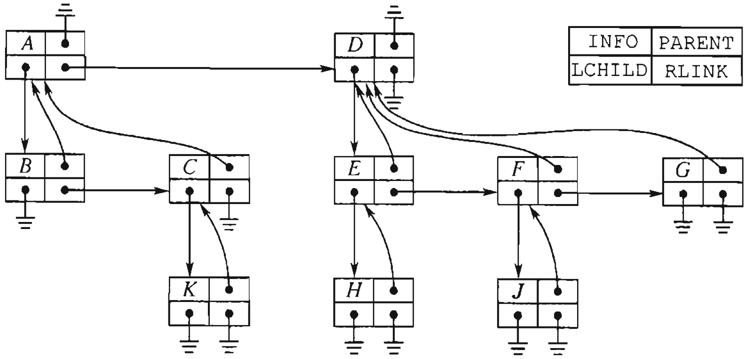


图 26 一根三重链接树

所有向上的链接就能画出树的框图来。除根之外每个节点都恰有一个父亲,但可能有好多儿子,所以比起给出向下的链接来,给出向上的链接更简单。但为什么在我们的讨论中,不更早些就考虑向上的链接呢?当然,答案是,向上的链接本身在大多数情况下几乎是不适当的,因为很难很快地说出一个节点是否终节点,或者给出它的任何儿子的地址,等等。然而,有一个非常重要的应用,其中向上链接本身就是充分的了。我们现在将简略地研究一下由 M. J. Fischer 和 B. A. Galler 给出的用于处理等价关系的优美的算法。

一个等价关系“$\equiv$”是对于对象集合 S 中的任何对象 x,y 和 z,满足以下三个性质的元素之间的关系:

i) 如果 $x\equiv y$ 且 $y\equiv z$,则 $x\equiv z$。(传递性。)

ii) 如果 $x\equiv y$,则 $y\equiv x$。(对称性。)

iii) $x\equiv x$。(反身性。)

(试把这个关系同 2.2.3 小节中的偏序关系的定义相比较,尽管三个定义性质中有两个相同,但是等价关系与偏序关系完全不同。)等价关系的例子有“=”关系,整数的同余(modulo m)关系,以及如 2.3.1 小节所定义的树之间的相似性关系,等等。

等价性的问题是成对地读入等价元素,而后基于给定的一些对,确定两个特定的元素能否证明是等价的。例如,假设 S 是集合{1,2,3,4,5,6,7,8,9},而且假设给了以下的配对:

$$1\equiv 5,\quad 6\equiv 8,\quad 7\equiv 2,\quad 9\equiv 8,\quad 3\equiv 7,\quad 4\equiv 2,\quad 9\equiv 3 \tag{11}$$

由此得出,例如,$2\equiv 6$,因为 $2\equiv 7\equiv 3\equiv 9\equiv 8\equiv 6$,但我们不能证明 $1\equiv 6$。事实上,(11)中的所有配对把 S 分成两类:

$$\{1,5\}\text{ 和 }\{2,3,4,6,7,8,9\} \tag{12}$$

使得两个元素等价,当且仅当它们属于相同的类。不难证明,任何等价关系把集合 S 划分成不相交的类(称为等价类),因此当且仅当两个元素属于同一个类时才能等价。

因此,等价问题的解就是记住像(12)这样的等价类的问题。我们可以从每个元素

单独成一类开始，于是：

$$\{1\}\ \{2\}\ \{3\}\ \{4\}\ \{5\}\ \{6\}\ \{7\}\ \{8\}\ \{9\} \qquad (13)$$

现在如果给出关系 $1\equiv5$，我们就可把{1,5}放在一个类中。在处理了头三个关系 $1\equiv5$，$6\equiv8$ 和 $7\equiv2$ 之后，可把(13)改成

$$\{1,5\}\quad \{2,7\}\quad \{3\}\quad \{4\}\quad \{6,8\}\quad \{9\} \qquad (14)$$

现在配对 $9\equiv8$ 把{6,8,9}放在一起，等等。

问题是要找出好的方法来在计算机内表示像(12)，(13)和(14)这样一些情况，使我们能有效地实施把一些类合并在一起的操作，以及测试两个给定的元素是否在相同的类中。为此目的以下的算法使用有向树结构：S 的元素变成一个有向森林的节点；而且作为迄今所读入的等价对的结果，两个节点等价，当且仅当它们属于同一棵树。这个测试是容易进行的，因为两个元素在同一棵树上，当且仅当它们是在同一个根元素之下。而且，通过简单地把有向树当做另一个的根的新子树，很容易就可把两个有向树合并在一起。

算法 E（处理等价关系）　令 S 是数{1,2,…,n}的集合，并令 PARENT[1]，PARENT[2]，…，PARENT[n]是整型变量。这个算法输入像(11)那样的一组关系，并且调整 PARENT 表来表示一个有向树的集合，使得作为给定关系的一个结果两个元素等价，当且仅当它们属于同一棵树。（注：在更一般的情况下，S 的元素是符号名而不是简单地为从 1 到 n 的数；然后像在第 6 章中那样，查找程序将确定对应于 S 的元素的节点的地址，而且 PARENT 将是在每个节点中的一个字段。对于这个更一般情况的修改是直截了当的。）

E1. [初始化] 对于 $1\le k\le n$，置 PARENT[k]←0。（这意味着，开始时所有树都只由一个根组成，像在(13)中那样。）

E2. [输入新的对] 从输入中得到下一对等价元素“$j\equiv k$”。若输入穷尽，算法结束。

E3. [寻找根] 如果 PARENT[j] > 0，则置 j←PARENT[j]，并重复此步。如果 PARENT[k] > 0，置 k←PARENT[k]，并重复此步。（在此操作之后，j 和 k 已经被移动到要使其等价的两棵树的根。输入关系 $j\equiv k$ 是多余的，当且仅当我们现在有 $j=k$。）

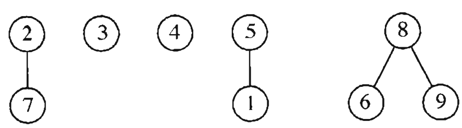
E4. [合并树] 如果 $j\ne k$，置 PARENT[j]←k，转回到步骤 E2。▌

读者应当对输入(11)尝试此算法。在处理了 $1\equiv5$，$6\equiv8$，$7\equiv2$ 和 $9\equiv8$ 之后，将有

PARENT[k]:	5	0	0	0	0	8	2	0	8
k:	1	2	3	4	5	6	7	8	9

(15)

它表示树



(16)

在这之后,(11)剩下的关系是更有趣的,请见习题 9。

在许多应用中都有这个等价问题出现。我们将在 7.4.1 小节研究图的连通性时,讨论对算法 E 的重大改进。当编译程序处理 FORTRAN 这样的语言中的“等价性声明”时,就会出现这个问题的更一般情况,它在习题 11 中讨论。

在计算机内存中表示树还有许多方法。回想一下在 2.2 节中我们讨论表示线性表的三个主要方法:带有终端链接 Λ 的直接表示,循环链接表以及双重链接表。在 2.3.1 小节中描述的无穿线二叉树表示在 LLINK 和 RLINK 两方面对应于一个直接表示。通过在 LLINK 和 RLINK 方向独立地使用这三个方法的任何一个，有可能得到八个其它二叉树表示。例如，图 27 示出，如果在两个方向上都使用循环链接，我们得到的情况。如果像在这个图当中那样，彻底地使用循环链接，我们有所谓的一个*环形结构*;在一些应用中已经证明环形结构是十分灵活的。像通常一样，表示的适当选择，依赖于对这些结构进行操作的算法中所需要的插入、删除和遍历的类型。已经考察过了迄今为止在这一章中所给出的例子的读者，应当毫不困难地理解如何处理任何这些存储表示。

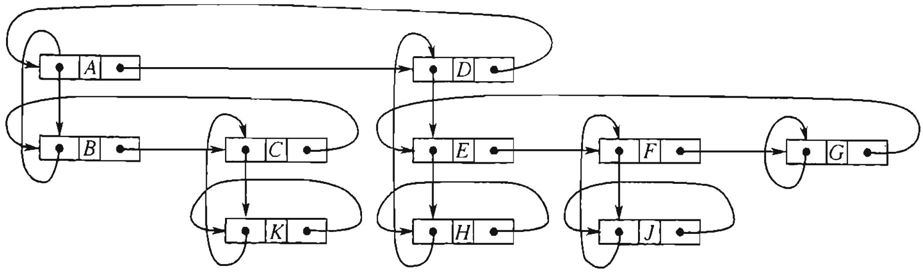


图 27　一个环形结构

我们以修改的双重链接环形结构应用于以前已经考虑过的一个问题,即对于多项式的算术运算,来结束本小节:假定两个多项式被表示为循环表,算法 2.2.4A 实施一个多项式同另一个多项式的相加。在该小节中的其它算法给出多项式的其它操作。然而,2.2.4 小节中的多项式被限定为至多有三个变量。当涉及多变量的多项式时,通常使用树结构而不是线性表是更适当的。

多项式既可是常数,也可具有形式

$$\sum_{0\le j\le n} g_j x^{e_j}$$

其中 x 是一个变量,$n>0$,$0=e_0<e_1<\cdots<e_n$,而且 $g_0,\cdots,g_n$ 是仅涉及按照字母顺序小于 x 的那些变量的多项式,$g_1,\cdots,g_n$ 不为零。多项式的这一递归定义本身就适合于如图 28 中所示的树的表示。节点有六个字段,在 MIX 的情况下它可以装入三个字中:

<table>
<tr><td>+</td><td>0</td><td>LEFT</td><td>RIGHT</td></tr>
<tr><td>+</td><td>EXP</td><td>UP</td><td>DOWN</td></tr>
<tr><td colspan="4">CV</td></tr>
</table>

(17)

这里 LEFT,RIGHT,UP 和 DOWN 是链接;EXP 是表示乘幂的整数;而 CV 既可是常数(系数)也可是变量的字符名。根节点有 UP = Λ,EXP = 0,LEFT = RIGHT = *(本身)。

下列算法说明在这种有四个方向的链接的树中的遍历、插入和删除,所以值得仔细加以研究。

算法 A(*多项式加法*)　假定 P 和 Q 是指针变量,链接到有图 28 所示形式的不同多项式树的根,本算法把多项式(P)加到多项式(Q)上。在本算法结束时,多项式(P)将不变,而多项式(Q)将包含两个多项式之和。

A1. [测试多项式的类型] 如果 DOWN(P) = Λ(即如果 P 指向一个常数),则置 Q←DOWN(Q)零次或多次直到 DOWN(Q) = Λ 为止并转到 A3。如果 DOWN(P) ≠ Λ,则如果 DOWN(Q) = Λ 或者如果 CV(Q) < CV(P),则转到 A2。否则如果 CV(Q) = CV(P),则置 P←DOWN(P),Q←DOWN(Q)并重复这一步骤;如果 CV(Q) > CV(P),则置 Q←DOWN(Q),并重复这一步骤。(步骤 A1 既可找到多项式的两个匹配的项,也可另外确定是否必须插入一个新变量到多项式(Q)的当前部分。)

A2. [向下插入] 置 R⇐AVAIL,S←DOWN(Q)。如果 S ≠ Λ,则置 UP(S)←R,S←RIGHT(S),而且如果 EXP(S) ≠ 0,重复此操作直到最终地 EXP(S) = 0 为止。置 UP(R)←Q,DOWN(R)←DOWN(Q),LEFT(R)←R,RIGHT(R)←R,CV(R)←CV(Q),以及 EXP(R)←0。最后,置 CV(Q)←CV(P) 以及 DOWN(Q)←R,并返回 A1。(我们已经在 NODE(Q)的紧底下插入一个"虚拟的"零多项式,以得到同在 P 的树内找到的对应多项式的一个匹配。在此步所完成的链接操作是直截了当的,因而如同在 2.2.3 小节所说明的那样,利用"前—后"框图,能很容易地推出。)

A3. [找到匹配] (这时 P 和 Q 指向给定的多项式对应的项,所以可容易地进行加法。)置 CV(Q)←CV(Q) + CV(P)。如果此和为零且如果 EXP(Q) ≠ 0,则转到步骤 A8。如果 EXP(Q) = 0,则转到步骤 A7。

A4. [向左推进] (在成功地完成一项的相加之后,寻找要相加的下一项。)置 P←LEFT(P)。如果 EXP(P) = 0,则转到 A6。否则置 Q←LEFT(Q)一次或多次直到 EXP(Q) ≤ EXP(P)为止。如果随后 EXP(Q) = EXP(P),则返回步骤 A1。

A5. [插到右边] 置 R⇐AVAIL。置 UP(R)←UP(Q),DOWN(R)←Λ,CV(R)←0,LEFT(R)←Q,RIGHT(R)←RIGHT(Q),LEFT(RIGHT(R))←R,RIGHT(Q)←R,EXP(R)←EXP(P),以及 Q←R。返回到步骤 A1。(我们需要在 NODE(Q)的紧右边,在当前的行中插入新的一项,以便来匹配在多项式(P)中对应的指数。和在步骤 A2 中一样,"前—后"框图可使这个操作更清楚。)

A6. [向上返回] (现在已经完全遍历多项式(P)的一行。)置 P←UP(P)。

A7. [Q 上移到正确的层次上] 如果 UP(P) = Λ,则转到步骤 A11;否则置 Q←UP(Q)

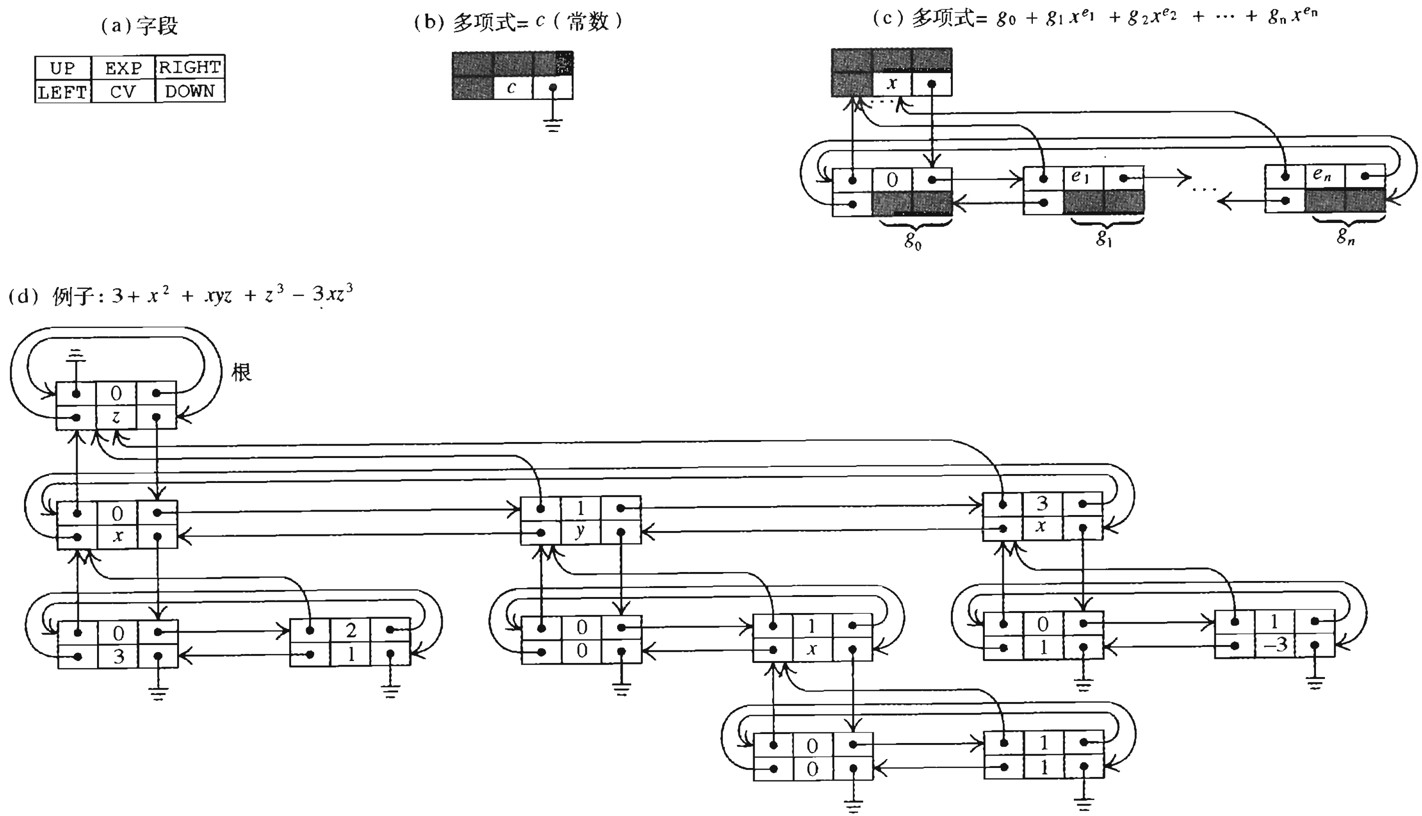


图 28　使用四个有向链接的多项式表示。节点的阴影区域表示同所考虑的上下文无关的信息

零次或多次直到 CV(UP(Q)) = CV(UP(P))为止，返回到步骤 A4。

A8. [删除为零的项] 置 R←Q，Q←RIGHT(R)，S←LEFT(R)，LEFT(Q)←S，RIGHT(S)←Q 以及 AVAIL⇐R。(出现消去，因此多项式(Q)的一个行元素被删除。)如果现在 EXP(LEFT(P)) = 0 和 Q = S，则转到步骤 A9，否则返回步骤 A4。

A9. [删除常数多项式] (消去已经引起多项式简化为常数，所以多项式(Q)的一行被删除。)置 R←Q，Q←UP(Q)，DOWN(Q)←DOWN(R)，CV(Q)←CV(R)以及 AVAIL ⇐R。置 S←DOWN(Q)；如果 S≠Λ，置 UP(S)←Q，S←RIGHT(S)，而且如果 EXP(S)≠0，则重复这一步骤直到最终 EXP(S) = 0 为止。

A10. [检测到零吗?] 如果 DOWN(Q) = Λ，CV(Q) = 0，以及 EXP(Q)≠0，则置 P←UP(P)，并转到步骤 A8，否则转到步骤 A6。

A11. [结束] 置 Q←UP(Q)零次或多次，直到 UP(Q) = Λ(于是把 Q 带到树的根处)。 ▌

如果多项式(P)有较少的项而多项式(Q)有许多项，则这个算法实际上将比算法 2.2.4A 运行得快得多，因为在加法的执行过程中不需要跑遍多项式(Q)的所有项。读者将发现用人工模拟算法 A，把多项式 $xy - x^2 - xyz - z^3 + 3xz^3$ 加到图 28 中所示的多项式上，是有教益的。(这一情况并不显示此算法的效率，但是通过显示必须加以处置的困难情况，它使算法跑遍所有步骤。)关于对算法 A 的进一步评述，请见习题 12 和 13。

这里我们不作这样的断言，即图 28 所示的表示对于多个变量的多项式是“最好的”。在第 8 章，我们将与使用辅助栈的算术算法一起，考虑多项式表示的另一种格式，以及当它同算法 A 作比较时，概念简洁的重要优点。我们对算法 A 的主要兴趣在于它表征了使用多个链接的对树的操作。

习　题

▶ 1.[*20*] 如果在像(8)这样的层次序的顺序表示中我们仅有 LTAG，INFO 和 RTAG 字段(而不是 LLINK)，是否有可能重新构造 LLINK？(换句话说，是否如同在(3)中的 RLINK 一样，(8)中的 LLINK 是多余的?)

2.[*22*] (Burks，Warren 和 Wright，*Math. Comp.* **8** (1954)，53～57) 以带度数的先根序存储的树(2)将是

DEGREE	2	0	1	0	3	1	0	1	0	0
INFO	*A*	*B*	*C*	*K*	*D*	*E*	*H*	*F*	*J*	*G*

[试和(9)作比较，在那里使用了后根序。] 试设计类似于算法 F 的算法，通过在此表示中从右到左地进行来计算节点的局部地定义的函数。

▶ 3.[*24*] 修改算法 2.3.2D 使得它遵循算法 F 的思想，但是它把它计算出来的导数作为中间结果放置到栈上，而不是像在步骤 D3 中那样以一种不规则的方式记录它们的地址。(请见习题 2.3.2－21。)栈可通过使用在每个导数的根处的 RLINK 字段来加以保持。

4.[*18*] 树(2)包含 10 个节点，其中 5 个是终节点。在正常的二叉树方式下这些树的表示涉

及 10 个 LLINK 字段和 10 个 RLINK 字段(每个节点 1 个)。在(10)那些树的表示中,LLINK 和 INFO 在一个节点中共享空间,它要求 5 个 LLINK 和 15 个 RLINK。在每种情况下共有 10 个 INFO 字段。

给定带有 n 个节点的一个森林,其中的 m 个是终端节点。试比较使用这两种树表示方法必须存储的 LLINK 和 RLINK 的总数。

5.[*16*] 如图 26 所示,一个三重链接树在每个节点中包含 PARENT,LCHILD 和 RLINK 字段。当在 PARENT,LCHILD 或 RLINK 字段中没有适当的节点来提及时自由地使用 Λ 链接。试问,通过如同在 2.3.1 小节所做的那样,放置"穿线"链接代替空的 LCHILD 和 RLINK 项,这样来把这个表示推广成穿线树,是否好的想法?

▶ 6.[*24*] 假设一个有向森林的节点有三个链接字段,即 PARENT,LCHILD 和 RLINK,但仅仅 PARENT 链接已被设置以表示树结构。每个节点的 LCHILD 字段是 Λ,而 RLINK 字段只作为以某种顺序把节点链接在一起的一个线性表而设置。链接变量 FIRST 指向头一个节点,而且最后节点有 RLINK = Λ。

试设计一个算法,它跑遍这些节点,并且利用 PARENT 链接,它填写 LCHILD 和 RLINK 字段,得到像图 26 中那样的三重链接树表示。而且还设置 FIRST 使得它现在指向在这个表示中的第一棵树的根。

7.[*15*] 如果在(11)中未给出关系 $9 \equiv 3$,在(12)中将出现什么类?

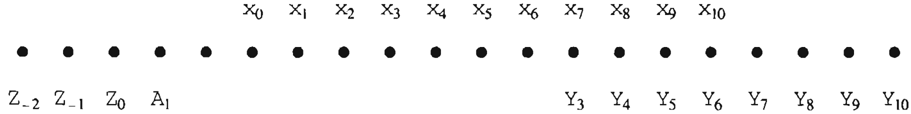
8.[*15*] 算法 E 建立一个树结构,表示给定的等价元素对,但正文中没有明确提到可如何使用算法 E 的结果。假定 $1 \leq j \leq n$, $1 \leq k \leq n$,并且假定对于某个等价集合,算法 E 已经建立 PARENT 表格,试设计一个算法,可回答"是否 $j \equiv k$?"的问题。

9.[*20*] 给出类似于(15)的一个表格和类似于(16)的一个框图,它在算法 E 从左到右地处理了(11)中的所有等价之后,示出存在的树。

10.[*28*] 在最坏的情况下,算法 E 可能花费数量级为 n^2 步的时间来处理 n 个等价。试说明如何修改这个算法使得最坏情况不致于这样坏。

▶ 11.[*24*] (*等价声明*) 好多种编译程序语言,著名的有 FORTRAN,提供了把重叠的内存单元分配给顺序存储的表格的工具。程序员把形如"X[j] ≡ Y[k]"的关系集合给了编译程序,这意味着对所有的 s,把变量 X[$j+s$]和变量 Y[$k+s$]分配到相同的单元。对于每个变量,也给出允许的下标的范围:"ARRAY X[$l:u$]" 意味着对于表格的表项 X[l],X[$l+1$],…,X[u]在内存中设置了用于存放它们的空间。对于每个变量的等价类,编译程序保留尽可能小的连续的内存单元块,来包含对于这些变量的允许的下标值的所有表项。

例如,假设我们有 ARRAY X[0:10],ARRAY Y[3:10],ARRAY A[1:1]以及 ARRAY Z[−2:0],加上等价关系 X[7] ≡ Y[3],Z[0] ≡ A[0],以及 Y[1] ≡ A[8]。我们必须为这些变量留出 20 个连续的单元



(A[1]之后的单元对于任何数组来说都是不允许的下标值,但无论如何必须加以保留。)

本题的目的是修改算法 E,使得它可应用于比刚才描述的更为一般的情况。假定我们正在为这种语言编写编译程序,而且在编译程序的程序本身内的表格就有为每个数组准备的一个节点,包含 NAME,PARENT,DELTA,LBD 和 UBD 字段。假定编译程序的程序以前已经处理了所有的

ARRAY 声明，使得如果“ARRAY X[l:u]”已经出现过，而且如果 P 指向 X 的节点，则

$$\text{NAME(P)} = \text{“X”},\quad \text{PARENT(P)} = \Lambda,\quad \text{DELTA(P)} = 0, \text{LBD(P)} = l,\quad \text{UBD(P)} = u$$

问题是要设计处理等价声明的算法，使得在实施了这个算法之后，

PARENT(P) = Λ 意味着单元 X[LBD(P)]，…，X[UBD(P)]在内存中被保留给这个等价类；

PARENT(P) = Q≠Λ 意味着单元 X[k]等于单元 Y[k + DELTA(P)]，其中 NAME(Q) = “Y”。

例如，在上面列出的等价之前我们可能有以下节点

P	NAME(P)	PARENT(P)	DELTA(P)	LBD(P)	UBD(P)
α	X	Λ	0	0	10
β	Y	Λ	0	3	10
γ	A	Λ	0	1	1
δ	Z	Λ	0	-2	0

在处理了等价之后，诸节点可能如下：

α	X	Λ	*	-5	14
β	Y	α	4	*	*
γ	A	δ	0	*	*
δ	Z	α	-3	*	*

（* 表示无关信息。）

试设计进行这一转换的算法。假定对你的算法的输入有(P，j，Q，k)的形式，表示“X[j]≡Y[k]”，其中 NAME(P) = “X”和 NAME(Q) = “Y”。应确保检查等价是否矛盾；例如，X[1]≡Y[2]和 X[2]≡Y[1]矛盾。

12.[*21*] 在算法 A 开始时，变量 P 和 Q 指向两棵树的根。令 P_0 和 Q_0 表示在算法 A 执行之前的 P 和 Q 的值。(a)在算法结束后，Q_0 是否总是两个给定多项式的和的根之地址？(b)在算法结束时，P 和 Q 恢复成它们的初始值 P_0 和 Q_0 吗？

▶ 13.[*M29*] 试给出下列事实的一个非形式证明，即在算法 A 的步骤 A8 开始时，我们总有 EXP(P) = EXP(Q)和 CV(UP(P)) = CV(UP(Q))。（这一事实对适当理解该算法很重要。）

14.[*40*] 给出算法 A 的正确性的一个形式化证明(或否定)。

15.[*40*] 试设计计算图 28 中那样表示的两个多项式乘积的算法。

16.[*M24*] 试证明算法 F 的正确性。

▶ 17.[*25*] 算法 F 计算“由底向上”局部地定义的函数，即在某个节点处计算函数之前，应先在该节点的儿子处进行计算。一个“由顶向下”局部地定义的函数是这样一种函数，即在一个节点 x 处 f 的值仅依赖于 x 以及在 x 的父亲处 f 的值。利用辅助栈，试设计类似于算法 F 的算法，它在树的每个节点处计算“由顶向下”的函数 f。（和算法 F 类似，你的算法应当对于以带有度数的后根序存储的树，像在(9)中那样，能够有效地工作。）

▶ 18.[*28*] 试设计一个算法，对于 $1 \leq j \leq n$，给定对应于先根序的顺序表示的两个表格 INFO1[j]和RLINK[j]，这个算法构造出对于 $1 \leq j \leq n$ 的表格 INFO2[j]和 DEGREE[j]，它们对应于带有度数的后根序。例如，按照(3)和(9)，你的算法应当把

j	1	2	3	4	5	6	7	8	9	10
INFO1[j]	A	B	C	K	D	E	H	F	J	G
RLINK[j]	5	3	0	0	0	8	0	10	0	0

转换成

INFO2[j]	B	K	C	A	H	E	J	F	G	D
DEGREE[j]	0	0	1	2	0	1	0	1	0	3

19.[*M27*] 代替使用(5)中的 SCOPE 链接,可以简单地以先根序列出每个节点的后裔数:

DESC	3	0	1	0	5	1	0	1	0	0
INFO	A	B	C	K	D	E	H	F	J	G

令 $d_1 d_2 \cdots d_n$ 是以这个方法得到的一个森林的后裔数序列。

a) 证明对于 $1 \leq k \leq n$,$k + d_k \leq n$,并证明 $k \leq j \leq k + d_k$ 意味着 $j + d_j \leq k + d_k$。

b) 反过来,证明若 $d_1 d_2 \cdots d_n$ 是满足 a)的条件的非负整数序列,则它是森林的后裔数序列。

c) 假设 $d_1 d_2 \cdots d_n$ 和 $d_1' d_2' \cdots d_n'$ 是两个森林的后裔数序列。证明有第三个森林存在,其后裔数是 $\min(d_1, d_1')\min(d_2, d_2')\cdots\min(d_n, d_n')$。

2.3.4 树的基本数学性质

树结构在计算机发明之前许多年就已作为广泛的数学研究对象了,而且关于它们已经发现了许多有趣的事实。在本小节里,我们将综述树的数学理论,它不仅为我们提供对于树结构的更多洞察,而且对于计算机的算法也有重要的应用。

建议非主修数学的读者跳到 2.3.4.5 小节,那里是我们后面要研究的在应用中经常出现的若干课题。

以下的内容大部分来自称为图论的一个更大的数学领域。不幸的是,在这个领域中大概从未有过标准的术语。因此作者遵循关于图论的现代著述的做法,即使用同关于图论的任何其它的书相似而又不完全相同的术语。在以下几个小节里(事实上,是在全本书里)我们试图从那些相当普遍地使用的而和其它普通的术语不形成强烈冲突的词汇中,选择对于重要概念的简短和描述性的词汇。在这里所使用的名称也偏向于计算机的应用。因此电气工程师可能更喜欢将我们所称的“自由树”称为“树”;但是我们要更短的术语“树”代表在计算机的文献中一般地使用的概念,而且在计算机应用中它是重要得多的概念。如果我们遵循关于图论的某些作者的术语,那我们就要说“有限的带标号的有根有序树”而不是简单的“树”,并说“拓扑分支的树木”而不是“二叉树”!

2.3.4.1 自由树 一般地把图定义为点(称为*顶点*)的集合连同连接某些不同的顶点对的线(称做*边*)的集合。至多有一个边连接任何一对顶点。如果有一条边连接两个顶点,则说这两个顶点是*相邻的*。若 V 和 V' 是顶点,$n \geq 0$,如果 $V = V_0$,对于 $0 \leq k < n$,V_k 相邻于 V_{k+1},而且 $V_n = V'$,我们说$(V_0, V_1, \cdots, V_n)$是从 V 到 V' 的长度为 n 的一条*通路*。如果 $V_0, V_1, \cdots, V_{n-1}$都不相同,而且 $V_1, V_2, \cdots, V_{n-1}, V_n$ 也不相同,则这条通路是*简单的*。如果图的任何两个顶点之间都有一条通路,则这个图是*连通的*。*回路*是从一个顶点到它自身长度为 3 或更大的一条简单通路。

图 29 说明了这些定义,它示出有五个顶点和六条边的一个连通图。顶点 C 同 A 相邻,但和 B 不相邻;从 B 到 C 有两条长度为 2 的通路,即(B, A, C)和(B, D, C)。回路有好几条,包括(B, D, E, B)在内。

*自由树*或“无根树”(图 30)定义为无回路的连通图。这个定义既适用于无穷图也

适用于有限图,尽管对于计算机应用说来,我们自然最关心有限树。定义自由树有许多等价的方法,其中一些方法出现在以下有名的定理中:

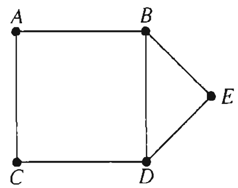


图 29 一个图

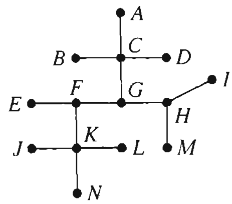


图 30 一个自由树

定理 A *如果 G 是一个图,则下列的诸命题等价:*

a) *G 是自由树。*

b) *G 是连通的,但若删去任何一个边,得到的图就不再是连通的。*

c) *如果 V 和 V' 是 G 的不同顶点,则从 V 到 V' 恰有一条简单通路。*

而且,如果 G 是有限的,恰含 $n>0$ 个顶点,则下列命题也等价于 a),b)和 c):

d) *G 不含回路且有 $n-1$ 条边。*

e) *G 是连通的且有 $n-1$ 条边。*

证明 a)蕴涵 b),因为如果把边 $V—V'$ 删去而 G 仍然连通,则必有长度为 2 或更大的一条简单通路$(V, V_1, \cdots, V')$(参见习题 2),于是$(V, V_1, \cdots, V', V)$将是 G 中的一个回路。

b) 蕴涵 c),因为从 V 到 V' 至少有一条简单通路。而如果有两条这样的通路$(V, V_1, \cdots, V')$和$(V, V_1', \cdots, V')$,我们将找到使得 $V_k \neq V_k'$ 的最小的 k。删去边 $V_{k-1}—V_k$ 将不使图断开,因为从 V_{k-1} 到 V_k 仍然有一条不使用被删去的边的通路$(V_{k-1}, V_k', \cdots, V', \cdots, V_k)$。

c) 蕴涵 a),因为如果 G 包含一个回路$(V, V_1, \cdots, V)$,则从 V 到 V_1 有两条简单通路。

为了证明 d)和 e)也等价于 a),b)和 c),首先让我们证明一个辅助的结果:如果 G 是没有回路且至少有一条边的任何有限的图,则至少有一个顶点,它恰好和另一个顶点相邻。这是因为我们可以取某顶点 V_1 和一个相邻顶点 V_2;对于 $k \geq 2$,或者 V_k 同 V_{k-1} 相邻而不和其它的相邻,或者它同一个我们称之为 $V_{k+1} \neq V_{k-1}$ 的顶点相邻。由于没有回路,$V_1, V_2, \cdots, V_{k+1}$ 必定是不同的顶点,所以这个过程最终必然终止。

现在假定 G 是有 $n>1$ 个顶点的自由树,并令 V_n 是仅仅相邻于另一顶点即 V_{n-1} 的顶点。如果我们删除 V_n 和边 $V_{n-1}—V_n$,则剩余的图 G' 是一棵自由树,因为 V_n 出现在 G 的非简单通路上,除非作为第一个或最后一个元素。这个论证(通过对 n 的归纳法)证明 G 有 $n-1$ 条边,因此 a)蕴涵 d)。

假定 G 满足 d)且设 V_n, V_{n-1}, G' 如同在上一段落中那样。于是图 G 是连通的,因为 V_n 被连接到 V_{n-1} 上,(由对 n 的归纳法)它被连接到 G' 的所有其它顶点上。因此 d)蕴涵 e)。

最后假定 G 满足 e)。如果 G 包含一个回路，则我们可以删除出现在该回路中的任何边，而 G 仍将是连通的。因此我们可以以这种方法继续删除边直到我们得到 $n-1-k$ 条边的且无回路的连通图 G' 为止。但由于 a)蕴涵 d)，我们必定有 $k=0$，即 $G=G'$。▌

自由树的概念可以直接地应用于计算机算法的分析中。在 1.3.3 小节中，我们讨论了基尔霍夫第一定律应用于计算一个算法的每一步被执行次数的问题；我们发现，基尔霍夫定律不完全确定每一步被执行的次数。但它减少了必须特别地解释的未知数的个数。树理论告诉我们还剩下多少独立的未知数，而且它给了我们寻求这些未知数的系统方法。

如果研究一个例子，理解下面的方法就会更容易些，因此我们在建立理论时，将对一个例子进行讨论。图 31 示出程序 1.3.3A 的一个抽象的流程图，它是受 1.3.3 小节中的基尔霍夫定律分析支配的。图 31 中的每个方框表示计算的一部分，而框里面的字母或数字表示在程序的一个流程中该计算将被执行的次数，这里使用 1.3.3 小节的记号。框与框之间的箭头表示程序中可能的转移。这些箭头都已标上标号 $e_1, e_2, \cdots, e_{27}$。我们的目的是找出由基尔霍夫定律所蕴涵的量 $A, B, C, D, E, F, G, H, J, K, L, P, Q, R$ 以及 S 之间的关系；同时我们希望能得到对于一般问题的某些启示。（注：在图 31 中已经作了某些简化；例如，C 和 E 之间的框已经标上“1”，而这事实上是基尔霍夫定律的一个结果。）

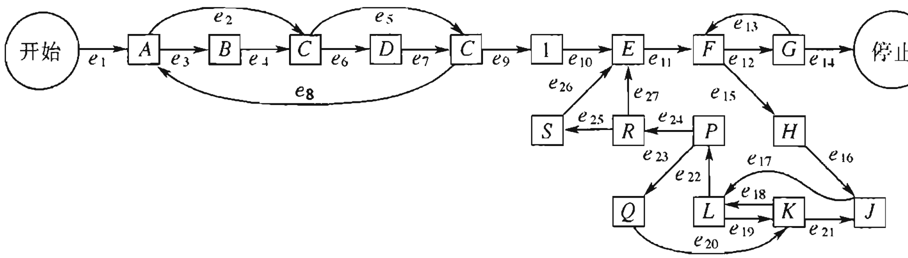


图 31　程序 1.3.3A 的抽象流程图

设 E_j 是被研究的程序执行分支 e_j 的次数；基尔霍夫定律是

$$\text{进入框的所有 } E \text{ 之和} = \text{框中的值} = \text{离开方框的所有 } E \text{ 值之和} \tag{1}$$

例如，在标记有 K 的方框的情况下，我们有

$$E_{19} + E_{20} = K = E_{18} + E_{21} \tag{2}$$

在以下的讨论中，代替 $A, B, \cdots, S$，我们将认为 $E_1, E_2, \cdots, E_{29}$ 是未知的。

图 31 的流程图可以进一步抽象化使得它变成如图 32 中那样的图 G。方框已经缩小成顶点，而箭头，$e_1, e_2, \cdots$ 现在表示图的边。（严格地说，图的边上没有隐含的方向，因而当我们谈及 G 的图论性质时，应忽略箭头的方向。然而，我们对于基尔霍夫定律的应用利用了箭头，后面我们很快就会看到。）为方便起见，从停止顶点到开始顶点，已

经画了额外的边 e_0,使得基尔霍夫定律可一致地应用于图的所有部分。图 32 也还包括对图 31 的其它一些小的改动;为把 e_{13}分成两个部分 e'_{13} 和 e''_{13},增加了一个额外的顶点和额外的边,使得图的基本定义(无两个边连接相同的两个顶点)保持正确;e_{19}也以这种方式被分开。如果我们有引向它本身的箭头的任何顶点,都要对之作相似的修改。

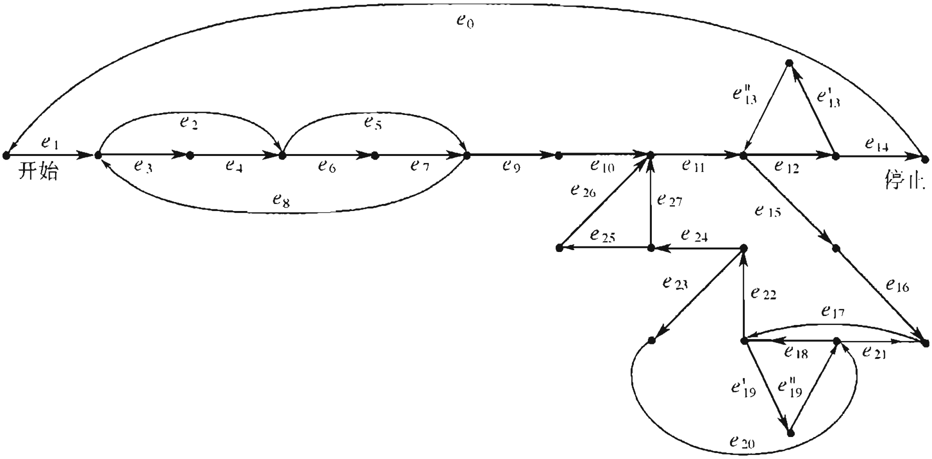


图 32　对应于图 31 的图,包括自由子树在内

在图 32 中有些边画得比其它边要粗些。这些边形成连接所有顶点的图的自由子树。总是有可能找出由流程图引出的图的自由子树,因为这些图必定是连通的,而且由定理 A 的 b),如果 G 连通而不是自由树,我们可以删除某条边而仍然得到连通的图;这个过程可以迭代进行直到我们到达自由子树为止。习题 6 是寻找自由子树的另一个算法。事实上,我们总是可以首先去掉边 e_0(它从停止顶点通到开始顶点);因此我们可以假定在所选定的子树中 e_0 不出现。

令 G' 是以这种方法找到的图 G 的自由子树,并且考虑不在 G' 中的 G 的任何边 $V—V'$。我们现在说明定理 A 的一个重要的结果:G' 加上这个新边包含一个回路;而且事实上,恰有一个回路,并有$(V, V', \cdots, V)$的形式,因为在 G' 中从 V' 到 V 有惟一的简单通路。例如,如果 G' 是图 32 中所示的自由子树,而且如果我们加上边 e_2,就得到沿着 e_2 进行而后(以同箭头相反的方向)沿 e_4 和 e_3 的一个回路,使用加号和减号来表示回路是否沿着箭头的方向,就可把这个回路代数地写作"$e_2 - e_4 - e_3$"。

如果我们对不在自由子树中的每条边实施这一过程,就可得到所谓的基本回路,在图 32 的情况下是

$$C_0:\quad e_0 + e_1 + e_3 + e_4 + e_6 + e_7 + e_9 + e_{10} + e_{11} + e_{12} + e_{14}$$

$$C_2:\quad e_2 - e_4 - e_3$$

$$C_5:\quad e_5 - e_7 - e_6$$

$$
\begin{aligned}
&C_8: && e_8 + e_3 + e_4 + e_6 + e_7 \\
&C''_{13}: && e''_{13} + e_{12} + e'_{13} \\
&C_{17}: && e_{17} + e_{22} + e_{24} + e_{27} + e_{11} + e_{15} + e_{16} \\
&C''_{19}: && e''_{19} + e_{18} + e'_{19} \\
&C_{20}: && e_{20} + e_{18} + e_{22} + e_{23} \\
&C_{21}: && e_{21} - e_{16} - e_{15} - e_{11} - e_{27} - e_{24} - e_{22} - e_{18} \\
&C_{25}: && e_{25} + e_{26} - e_{27}
\end{aligned}
\tag{3}
$$

显然,不在自由子树中的边 e_j 将仅出现于基本回路之一上,即 C_j 上。

我们现在正在走向这一构造的高峰。每个基本回路表示基尔霍夫方程的一个解;例如,对应于 C_2 的解是令 $E_2 = +1, E_4 = -1, E_3 = -1$,而其余的 $E = 0$。显然沿着图中的一个回路的流程总是满足基尔霍夫定律的条件(1)。而且,基尔霍夫方程是"齐次的",所以(1)的解的和与差产生另一个解。因此,我们可以得出结论,在下面的意义下,$E_0, E_2, E_5, \cdots, E_{25}$的值是独立的:

如果 $x_0, x_2, \cdots, x_{25}$*是任何实数*(对于每个不在子树 G' 中的 e_j 都有一个 x_j),
则基尔霍夫方程(1)*有一个解,使得* $E_0 = x_0, E_2 = x_2, \cdots, E_{25} = x_{25}$ (4)

这样一个解是通过围绕回路 C_0 走 x_0 次,围绕回路 C_2 走 x_2 次,等等求得的。而且,我们发现,剩余的变量 $E_1, E_3, E_4, \cdots$的值完全依赖于值 $E_0, E_2, \cdots, E_{25}$:

在命题(4)*中提到的解是惟一的* (5)

因为如果基尔霍夫方程有两个解使得 $E_0 = x_0, \cdots, E_{25} = x_{25}$,我们可以从一个解减去另一个解,因此我们得到其中有 $E_0 = E_2 = E_5 = \cdots = E_{25} = 0$ 的一个解。但是现在所有的 E_j 都必须为零,因为我们容易看到,当图是自由树时,基尔霍夫方程的非空解是不可能的(请见习题 4)。因此两个被假定的解必须相等。我们现在已经证明,基尔霍夫方程的所有解可以通过基本回路的乘法的和得到。

当这些说明被应用于图 32 的图中时,借助于独立变量 $E_0, E_2, \cdots, E_{25}$,我们得到基尔霍夫方程的通解:

$$
\begin{aligned}
&E_1 = E_0 && E_{14} = E_0 \\
&E_3 = E_0 - E_2 + E_8 && E_{15} = E_{17} - E_{21} \\
&E_4 = E_0 - E_2 + E_8 && E_{16} = E_{17} - E_{21} \\
&E_6 = E_0 - E_5 + E_8 && E_{18} = E''_{19} + E_{20} - E_{21} \\
&E_7 = E_0 - E_5 + E_8 && E'_{19} = E''_{19} \\
&E_9 = E_0 && E_{22} = E_{17} + E_{20} - E_{21} \\
&E_{10} = E_0 && E_{23} = E_{20} \\
&E_{11} = E_0 + E_{17} - E_{21} && E_{24} = E_{17} - E_{21} \\
&E_{12} = E_0 + E''_{13} && E_{26} = E_{25}
\end{aligned}
\tag{6}
$$

$E'_{13} = E''_{13}$　　　　　　　　　　$E_{27} = E_{17} - E_{21} - E_{25}$

为得到这些方程，对于在子树中的每个边 e_j，我们仅仅以适当的符号列出在回路 C_k 中出现 E_j 的所有 E_k。[因此(6)中的系数矩阵仅仅是(3)中系数矩阵的转置。]

严格地说，C_0 不应叫做基本回路，因为它涉及特殊的边 e_0。我们可以称 C_0 减边 e_0 是从开始到停止的基本通路。我们的边界条件，即在流程图中的开始和停止框恰好被执行一次，等价于关系

$$E_0 = 1 \tag{7}$$

前边的讨论说明怎样得到基尔霍夫定律的所有解；同一方法可以应用于(如同基尔霍夫本人应用它一样)电器线路而不是程序流程图。这时自然要问，基尔霍夫定律对于程序流程图的情况是否给出了最强可能的方程组，或者更可以说：从开始进行到停止的计算机程序的任何执行给了我们对于每个边被遍历次数的值 $E_1, E_2, \cdots, E_{27}$ 的一个集合，而且这些值遵守基尔霍夫定律；但是否有这样的基尔霍夫方程的解，它们不对应于任何计算机程序的执行呢？(在这个问题中，我们并不假定已了解给定的计算机程序，除开它的流程图外。)如果有这样的解存在，它们满足基尔霍夫的条件，但不对应于实际的程序执行，我们可以给出比基尔霍夫定律更强的条件。对于电器线路的情况，基尔霍夫本人给出第二定律[*Ann. Physiks und Chemie* **64** (1845), 497 ~ 514]：围绕一个基本回路电压降落之和必须为零。这第二定律不适用于我们的问题。

确实有明显的进一步的 E 必须满足的条件，如果它们要对应从开始到停止的流程图中某个实际的通路的话；它们必须是整数，而且事实上，它们必须是非负整数。这不是一个平凡的条件，因为我们不能简单地把任何任意的非负整数的值赋给独立变量 $E_2, E_5, \cdots, E_{25}$；例如，如果我们取 $E_2 = 2$ 和 $E_8 = 0$，从(6)和(7)我们求得 $E_3 = -1$。(因此，如果不通过分支 e_8 至少一次，图 31 中的流程图就没有任何执行两次采用分支 e_2。)所有的 E 都是非负整数这个条件也不充分；例如，考虑其中 $E''_{19} = 1, E_2 = E_5 = \cdots = E_{17} = E_{20} = E_{21} = E_{25} = 0$ 的解；没有办法到达 e_{18}，除非通过 e_{15}。下列条件是回答在上一段中提出的问题的充分必要条件：令 $E_2, E_5, \cdots, E_{25}$ 是任何给定的值，并且按照(6)，(7)确定 $E_1, E_3, \cdots, E_{27}$。假定所有的 E 都是非负整数，如果由满足 $E_j > 0$ 的那些边 e_j 以及触及这些边的顶点构成一个图，则它是连通的。于是从开始到停止有一条通路，其中每个边 e_j 恰被遍历 E_j 次。这一事实在下一小节中证明(请见习题 2.3.4.2-24)。

现在让我们来总结上面的讨论：

定理 K　*如果流程图(如图 31 那样)包含 n 个方框(包括开始和停止在内)以及 m 个箭头，有可能找出 $m - n + 1$ 个基本回路和从开始到停止的一条基本通路，使得从开始到停止的任何通路是等价于(就每个边被遍历的次数而言)基本通路的一次遍历加上每个基本回路惟一确定的遍历次数*。(基本通路和基本回路可能包括与边上的箭头所示方向相反的某些边的遍历；我们约定地说，这样的边被遍历 -1 次。)

反之，对于基本通路和基本回路的任何这样的遍历，即其中，每个边被遍历的总共的次数是非负的，且其中对应于一个正次数的遍历的顶点和边形成一个连通图，则至少

有一条从开始到停止的等价通路。 ▎

基本回路是通过挑出像在图 32 中那样的子树找到的；如果我们选择所得到的一个不同的子树，一般说来，我们就得到不同的基本回路集合。有 $m-n+1$ 个基本回路这一事实是从定理 A 得到的。我们从图 31 得到图 32 所作的修改，在添加了 e_0 之后，并不改变 $m-n+1$ 的值，尽管它们可能使 m 和 n 都增加值；这个构造已被推广以便完全地避免这些平凡的修改(请见习题 9)。

定理 K 是令人鼓舞的，因为它说，基尔霍夫定律(由 m 个未知数 $E_1, E_2, \cdots, E_m$ 组成 n 个方程)刚好有一个“冗余”：这 n 个方程允许我们消去 $n-1$ 个未知数。然而，在整个讨论中未知变量已经是边被遍历的次数，而不是进入流程图的每个方框的次数。习题 8 说明怎样来构造边对应于流程图的方框的另一个图，使得上面的理论可以用来推导感兴趣的变量之间真正的冗余数。

Thomas Ball 和 James R. Larus 在*ACM Trans. Prog. Languages and Systems* **16** (1994), 1319 ~ 1360 上讨论了定理 K 在用于测量高级语言程序性能的软件方面的应用。

习 题

1. [*14*] 列出在图 29 的图中出现的从 B 到 B 的所有回路。

2. [*M20*] 试证明，如果 V 和 V' 是一个图的顶点而且从 V 到 V' 有一条通路，则从 V 到 V' 有一条简单通路。

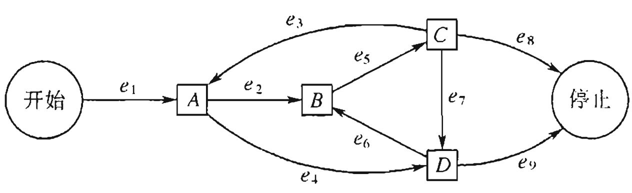
3. [15] 从开始到停止的什么通路等价于(在定理 K 的意义下)基本通路的遍历加上图 32 中回路 C_2 的一个遍历？

▶ 4. [*M20*] 设 G' 是一个有限自由树，其中箭头已画在它的边 $e_1, \cdots, e_{n-1}$ 上；设 $E_1, \cdots, E_{n-1}$ 是 G' 中满足基尔霍夫定律(1)的数。证明 $E_1 = \cdots = E_{n-1} = 0$。

5. [*20*] 使用等式(6)，借助于独立变量 $E_2, E_5, \cdots, E_{25}$ 来表达出现于图 31 中的框里边的量 $A, B, \cdots, S$。

▶ 6. [*M27*] 假设一个图有 n 个顶点 $V_1, \cdots, V_n$ 和 m 个边 $e_1, \cdots, e_m$。每条边 e 由一对整数(a, b)表示，如果它把 V_a 和 V_b 相连的话。试设计一个算法，它接受输入对(a_1, b_1), …, (a_m, b_m)，并打印出形成一个自由树的边的集合；如果这不可能，则算法报告失败。努力实现一个有效的算法。

7. [*22*] 利用由边 e_1, e_2, e_3, e_4, e_9 组成的自由子树，完成正文中对于流程图

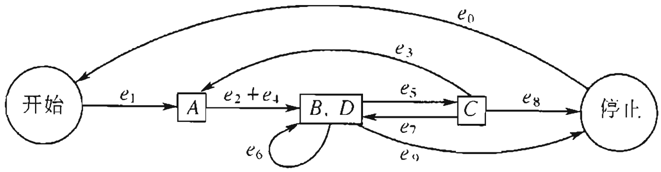


的构造。什么是基本回路？借助于 E_5, E_6, E_7 和 E_8 来表达 E_1, E_2, E_3, E_4, E_9。

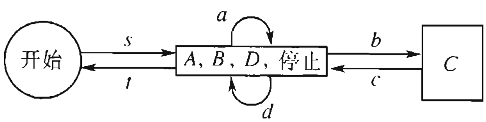
▶ 8. [*M25*] 当应用基尔霍夫第一定律于程序流程图时，我们通常仅仅对顶点流(即流程图的

每个框被执行的次数)感兴趣,而不是正文中所分析的边流。例如,在习题 7 的图中,顶点流是 $A = E_2 + E_4, B = E_5, C = E_3 + E_7 + E_8, D = E_6 + E_9$。

如果我们把某些顶点组合在一起,把它们处理为“超级顶点”,我们就可以把对应于相同顶点流的边流也组合起来。例如,如果在上面的流程图中我们把 B 和 D 组合在一起,则边 e_2 和边 e_4 可以组合在一起:



(如同在正文中一样,这里从停止到开始,也加上边 e_0。)继续这一过程,我们可以组合 $e_3 + e_7$,然后 $(e_3 + e_7) + e_8$,然后 $e_6 + e_9$,直到我们得到有边 $s = e_1, a = e_2 + e_4, b = e_5, c = e_3 + e_7 + e_8, d = e_6 + e_9, t = e_0$ 的既约流程图为止。在原来流程图中的每个顶点恰好有一条边:



由构造看出,在这个既约流程图中基尔霍夫定律成立。新的边流是原来的顶点流;因此,应用于既约流程图,正文中的分析说明原来的顶点流如何彼此依赖。

试证明,在下面的意义下,这一简化过程可以逆转,即在既约流程图中满足基尔霍夫定律的任何流的集合 $\{a, b, \cdots\}$ 可以被“分裂”成在原来的流程图中的边流的集合 $\{e_0, e_1, \cdots\}$。这些流 e_j 满足基尔霍夫定律,而且组合产生出给定的流 $\{a, b, \cdots\}$;然而,其中某些可能为负。(尽管这里仅仅对于一个特殊的流程图说明了简化过程,但你的证明应是一般地成立的。)

9. [*M22*] 在图 32 中边 e_{13} 和 e_{19} 被分裂成两部分,因为一个图不应有连接相同两个顶点的两个边。然而,如果我们察看这个构造的最后结果,分裂成两部分似乎很不自然,因为 $E'_{13} = E''_{13}$ 和 $E'_{19} = E''_{19}$ 是在(6)中发现的两个关系,而 E''_{13} 和 E''_{19} 是两个独立的变量。试说明如何能推广这一构造使得能避免边的不自然的分裂。

10. [*16*] 一位设计计算机的电路的电气工程师,有 n 个接线端 $T_1, T_2, \cdots, T_n$,它们应当在所有时间里具有实质上相同的电压。为实现这一点,这个工程师可以在任何一对接线端间焊上电线;想法是用足够多的电线连接使得从任何接线端到任何其它的接线端存在有通过电线的一条通路。证明为连接所有接线端所需要的最少电线数是 $n - 1$,而且 $n - 1$ 条电线实现所要求的连接,当且仅当它们形成自由树(且接线端和电线代表顶点和边)。

11. [*M27*] (R.C.Prim, *Bell System Tech. J.* **36** (1957), 1389 ~ 1401) 考虑习题 10 的电线连接问题,并附加以下假定,即对于每个 $i < j$,给定一个费用 $c(i, j)$ 表示连接接线端 T_i 到 T_j 的费用。试证明以下的算法给出一个最小费用的连接树:“如果 $n = 1$,什么也不做;否则,对接线端重新编号 $\{1, \cdots, n-1\}$ 并同费用挂钩使得 $c(n-1, n) = \min_{1 \le i < n} c(i, n)$;连接接线端 T_{n-1} 到 T_n 上;然后对于 $1 \le j < n-1$,把 $c(j, n-1)$ 变成 $\min(c(j, n-1), c(j, n))$,并且使用这些新的费用对于 $n-1$ 个接线端 $T_1, \cdots, T_{n-1}$ 重复执行此算法。(此算法在下述情形下被重复执行:每当随后要求在现在称为 T_j 和 T_{n-1} 的接线端之间进行连通时,如果在现在叫做 T_j 和 T_n 的接线端之间的连通更便

宜,则实际上就进行后一连通;因此 T_{n-1} 和 T_n 在算法的剩余部分被当做像是一个接线端一样。)"这个算法也可叙述如下:"从选定特定的接线端开始;然后从未选定的接线端到已选定的一个重复地作最便宜的连接,直到所有接线端都被选定为止。"

例如,考虑图 33(a),在栅格上它显示九个接线端;令连接两个接线端的费用是接线的长度,即它们之间的距离。(读者不妨用手算来尝试找出一个最小费用树,并且使用直观的方法而不是建议的算法。)这个算法将首先连接 T_8 到 T_9 上,然后 T_6 到 T_8,T_5 到 T_6,T_2 到 T_6,T_1 到 T_2,T_3 到 T_1,T_7 到 T_3,最后或者 T_4 到 T_2 或者 T_4 到 T_6。在图 33(b)中示出一个最小费用树(接线长度是 $7+2\sqrt{2}+2\sqrt{5}$)。

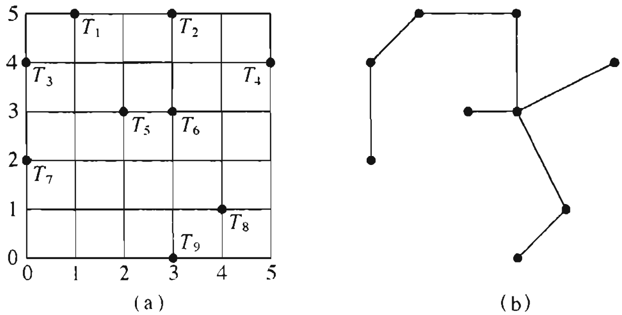


图 33 最小费用的自由树(见习题 11)

▶ 12.[29] 习题 11 的算法没有以适合于计算机的直接实现的方式来叙述。试重新阐述该算法,以计算机程序能以适当效率执行这个过程的方式,更详细地叙述要做的操作。

13.[M24] 在习题 6 的记号下,考虑有 n 个顶点和 m 个边的图。试证明有可能把整数 $\{1,2,\cdots,n\}$ 的任何排列写成为转置之乘积 $(a_{k_1}b_{k_1})(a_{k_2}b_{k_2})\cdots(a_{k_t}b_{k_t})$ 当且仅当该图是连通的。(因此有 $n-1$ 个转置的集合,它们生成 n 个元素的所有排列,但 $n-2$ 个转置的集合则不成。)

2.3.4.2 有向树 在上一小节中,我们看到如果忽略流程图边上箭头的方向,则抽象的流程图可看做图。回路、自由子树等图论的思想,被证明在流程图的研究中是相关的。当为每个边的方向赋予更大的意义时,则还有更多的话可说。而且在这种情况下,我们有了所谓的"有向图"。

让我们形式地把有向图定义为顶点和有向边的集合,每条有向边从顶点 V 通向顶点 V'。如果 e 是从 V 到 V' 的一个有向边,我们说 V 是 e 的*初始顶点*,而 V' 是*最终顶点*,而且我们写 $V=\text{init}(e)$,$V'=\text{fin}(e)$。不排除 $\text{init}(e)=\text{fin}(e)$ 的情况(尽管在通常的图中这一点从边的定义中被排除)。而且若干不同的有向边可以有相同的初始顶点和最终顶点。顶点 V 的*出度*是由它导出的有向边的条数,即是使得 $\text{init}(e)=V$ 的有向边 e 的条数;相似地,V 的*入度*被定义为具有 $\text{fin}(e)=V$ 的有向边的条数。

对于有向图的通路和回路的概念,是以和对普通的图的对应定义相类似的方式来加以定义的。但必须考虑某些重要的新术语。若 $e_1,e_2,\cdots,e_n$ 是有向边(且 $n\geq 1$),我们说 $(e_1,e_2,\cdots,e_n)$ 是从 V 到 V' 长度为 n 的一条*有向通路*,如果 $V=\text{init}(e_1)$,$V'=\text{fin}(e_n)$,而且对于 $1\leq k<n$,$\text{fin}(e_k)=\text{init}(e_{k+1})$ 的话。一条有向通路 $(e_1,e_2,\cdots,e_n)$ 说是

简单的，如果 $\mathrm{init}(e_1),\cdots,\mathrm{init}(e_n)$ 都不相同而且 $\mathrm{fin}(e_1),\cdots,\mathrm{fin}(e_n)$ 也不相同的话。有向回路是从一个顶点到它自身的简单有向通路。(有向回路可以有 1 或 2 的长度。但在上一小节中，这样短的回路从我们"回路"的定义中被排除。读者知道为什么这讲得通吗?)

作为这些直截了当的定义的例子，我们可以参照上一小节中的图 31。标号为"J"的框是入度为 2(由于有向边 e_{16}, e_{21} 的关系)和出度为 1 的顶点。序列 $(e_{17}, e_{19}, e_{18}, e_{22})$ 是从 J 到 P 长度为 4 的一条有向通路；这个通路不是简单的，因为例如，$\mathrm{init}(e_{19}) = L = \mathrm{init}(e_{22})$。这个框图不包含长度为 1 的回路，但 (e_{18}, e_{19}) 是长度为 2 的有向回路。

一个有向图，如果对于任何两个顶点 $V \neq V'$，从 V 到 V' 有一条有向通路存在，则称该有向图是*强连通的*；如果至少存在一个*根*，即至少有一个顶点 R 使得对于所有 $V \neq R$，存在从 V 到 R 的一条有向通路，则称该有向图是*有根的*。"强连通"总是意味着"有根的"，但反之并不成立。像在上一小节中的图 31 这样的流程图，是有根有向图的例子，且根顶点 R 是停止顶点；通过附加的从停止到开始的有向边(图 32)，它变成为强连通的。

如果我们忽略方向并且放弃重复的边或环，每个有向图 G 就以明显的方式对应于普通的图 G_0。形式地说，G_0 有从 V 到 V' 的一个边，当且仅当 $V \neq V'$，而且 G 有从 V 到 V' 或从 V' 到 V 的一个有向边。我们可以谈及 G 中(无向的)*通路*和*回路*，并且理解为，这些是 G_0 的通路和回路；我们可以说，G 是*连通的*——这是比"强连通"要弱得多的性质，甚至比"有根的"还要弱——如果对应的图 G_0 是连通的话。

其他作者有时把*有向树*(见图 34)叫做"有根树"，它是带有一个特定顶点 R 的有向图，使得

a)每个顶点 $V \neq R$，恰是一个有向边的初始顶点，该有向边用 $e[V]$ 表示。

b)R 不是任何有向边的初始顶点。

c)在上述定义的意义下 R 是根(即对于每个顶点 $V \neq R$，有从 V 到 R 的有向通路)。

立即可以得出，对于每个顶点 $V \neq R$，有从 V 到 R 的惟一的有向通路。因此，没有有向回路。

当有有限多个顶点时，容易看出，我们以前的"有向树"的定义和刚才给出的新定义是相兼容的。顶点对应于节点，有向边 $e[V]$ 是从 V 到 PARENT$[V]$ 的一个链接。

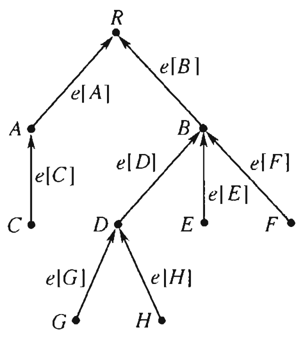


图 34　一棵有向树

由于性质 c)，与有向树对应的(无向)图是连通的。进一步说，此图还没有回路。因为，如果 $(V_0, V_1, \cdots, V_n)$ 是 $n \geqslant 3$ 的无向回路，而且如果 V_0 和 V_1 之间的边是 $e[V_1]$，则 V_1 和 V_2 之间的边是 $e[V_2]$，而且类似地，对于 $1 \leqslant k \leqslant n$，$V_{k-1}$ 和 V_k 之间的边一定是 $e[V_k]$，这同不存在回路相矛盾。如果 V_0 和 V_1 之间的边不是 $e[V_1]$，它必定是 $e[V_0]$，同样的论证必适用于回路

$$(V_1, V_0, V_{n-1}, \cdots, V_1)$$

因为 $V_n = V_0$。*所以当忽略有向边的方向时,有向树就是自由树*。

反之,重要的是注意到,我们可以逆转刚才描述的过程。如果我们以任何非空的自由树开始,例如像图30中那样,我们可以选择任何顶点作为根 R,并且对边赋予方向。直观的说法是在顶点 R 处"提起"图来并加以抖动,然后指定向上指的箭头。更形式地说,规则如下:

把 $V—V'$ 的边改成从 V 到 V' 的有向边,当且仅当从 V 到 R 的简单通路通过 V',即,如果它有 $(V_0, V_1, \cdots, V_n)$ 的形式,其中 $n>0, V_0 = V, V_1 = V', V_n = R$。

为验证这样的构造是正确的,我们需要证明每个 $V—V'$ 的边被赋予 $V \leftarrow V'$ 的方向或者 $V \rightarrow V'$ 的方向;而这是容易证明的,因为如果 $(V, V_1, \cdots, R)$ 和 $(V', V'_1, \cdots, R)$ 是简单通路,则除非 $V = V_1'$ 或 $V_1 = V'$,否则就有一个回路。这个构造揭示,如果我们知道哪一个顶点是根,则在有向树中有向边的方向就完全确定了。所以当根被明确地指出时,边的方向就不必示出了。

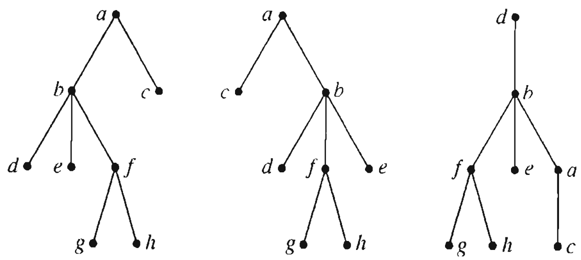


图35 三种树结构

我们现在看三种类型的树,即(有序的)树,如同在2.3节开始处定义的那样,它们在计算机程序中有重大的重要性,有向树(或无序树)以及自由树之间的关系。后面两种树在计算机算法研究中也出现,但不如第一种类型那么经常。*这些类型的树的结构之间的实质性区别仅仅是有关信息的数量*。例如,图35示出三种树,如果把它们当做有序树(其根在顶部)的话,它们是不同的。作为有向树,第一个和第二个是相同的,因为子树自左向右的顺序是无关紧要的。作为自由树,图35的所有三个图是相同的,因为根是无关紧要的。

在有向图中的*欧拉回路*是这样一种有向通路 $(e_1, e_2, \cdots, e_m)$,使得在有向图中的每一个边都恰出现一次,而且 $\mathrm{fin}(e_m) = \mathrm{init}(e_1)$。这是对于这个有向图的有向边"完全的遍历"。(欧拉回路是从欧拉于1736年关于在星期日的散步中不可能遍历柯尼斯堡的七座桥的每一座桥恰好一次的著名讨论中获名的。他还处理了无向图的类似问题。欧拉回路应同"哈密顿回路"相区别,后者是遇到每一个顶点恰一次的有向回路。请见第7章。)

如果有向图每个顶点的入度和出度相同,即,如果以 V 作为它们的初始顶点的边和以 V 作为它们最终顶点的边一样多,则称此有向图是*平衡的*(图36)。这个条件同基尔霍夫定律密切有关(见习题24)。如果有向图有一个欧拉回路,明显地它必是连通的而且是平衡的,除非它有*孤立的顶点*,孤立顶点是入度和出度都为零的顶点。

迄今为止，在本小节中，我们已经考察了不少的定义(有向图，有向边，初始顶点，最终顶点，出度，入度，有向通路，简单有向通路，有向回路，有向树，欧拉回路，孤立顶点，以及强连通，有根的和平衡的)，但只有很少的重要结果同这些概念相联系。现在我们已对介绍这些更丰富的材料做好了准备。一个基本的结果是由 I.J.Good 给出的一个定理[*J.London Math. Soc.* **21** (1947)，167～169]，他证明欧拉回路经常是可能的，除非它们明显地不可能：

定理 G　*没有孤立顶点的有限有向图具有欧拉回路，当且仅当它连通且平衡。*

证明　假定 G 是平衡的，且设

$$P = (e_1, \cdots, e_m)$$

是不两次使用一条有向边的最大长度的有向通路。于是，如果 $V = \text{fin}(e_m)$，而且如果 k 是 V 的出度，则有 $\text{init}(e) = V$ 的所有 k 个有向边必定都已在 P 中出现；否则我们可以加上 e 而得到一个更长的通路。但如果 $\text{init}(e_j) = V$ 以及 $j > 1$，则 $\text{fin}(e_{j-1}) = V$。因此，由于 G 是平衡的，我们必定有 $\text{init}(e_1) = V = \text{fin}(e_m)$，否则 V 的入度将至少为 $k+1$。

现在由 P 的循环排列得出，不在通路中的任何有向边 e 同在通路中的任何有向边不含有共同的初始顶点，也不含有共同的最终顶点。因此，如果 P 不是一个欧拉回路，G 就不连通。▌

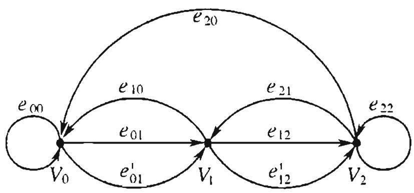


图 36　一个平衡的有向图

在欧拉回路和有向树之间存在重要的联系：

引理 E　*设* $(e_1, \cdots, e_m)$*是没有孤立顶点的有向图 G 的欧拉回路。设* $R = \text{fin}(e_m) = \text{init}(e_1)$。*对于每个顶点* $V \neq R$，*令* $e[V]$*是在这个回路中从* V *最后离开的；即*

$$e[V] = e_j, \text{当}\,\text{init}(e_j) = V\,\text{且}\,\text{init}(e_k) \neq V\ (j < k \leqslant m)\,\text{时} \quad (1)$$

则带有有向边 $e[V]$*的 G 的顶点形成带有根* R *的有向树。*

证明　有向树定义的性质 a)和 b)显然被满足。由习题 7，我们只需证明，在 $e[V]$ 当中没有有向回路。但这立即成立，因为如果 $\text{fin}(e[V]) = V' = \text{init}(e[V'])$，其中 $e[V] = e_j$ 和$e[V] = e_{j'}$，则 $j < j'$。▌

如果我们转而考虑对于每一个顶点的"第一次进入"，也许会更好地理解这个引理；第一次进入形成无序的树且所有的有向边都从 R 指向外。引理 E 有一个令人惊讶和重要的逆定理，是由 T.van Aardenne-Ehrenfest 和 N.G.de Bruijn 证明的[*Simon Stevin* **28**

(1951),203~217]:

定理 D *设 G 是一个有限的,平衡的有向图,且设 G' 是由 G 的顶点加上 G 的某些有向边组成的有向树。设 R 是 G' 的根,而且设 $e[V]$ 是初始顶点为 V 的 G' 的有向边。设 e_1 是有 $\mathrm{init}(e_1)=R$ 的 G 的任何有向边,则 $P=(e_1,e_2,\cdots,e_m)$ 是一个欧拉回路,如果该回路是有向通路而且对于它*

i)*没有有向边被使用一次以上,即当 $j\neq k$ 时,$e_j\neq e_k$。*

ii)*$e[V]$ 在 P 中不被使用,除非它是同规则* i)*相一致的惟一选择。即,如果 $e_j=e[V]$,而且如果 e 是具有 $\mathrm{init}(e)=V$ 的一个有向边,则对于某个 $k\leq j$,$e=e_k$。*

iii)*仅当 P 不能由规则* i)*再继续下去时,它才结束;即,如果 $\mathrm{init}(e)=\mathrm{fin}(e_m)$,则对于某个 k,$e=e_k$。*

证明 由 iii)和在定理 G 的证明中的论证,我们必定有 $\mathrm{fin}(e_m)=\mathrm{init}(e_1)=R$。现在如果 e 是不在 P 中出现的有向边,令 $V=\mathrm{fin}(e)$。由于 G 是平衡的,由此得出,V 是不在 P 中出现的某个有向边的初始顶点;而且如果 $V\neq R$,条件 ii)告诉我们,$e[V]$ 不在 P 中。现在对 $e=e[V]$ 使用同样的论证,因而我们最终发现 R 是不在通路中出现的某个有向边的初始顶点,这同 iii)矛盾。▎

定理 D 的实质是,给定图的任何有向子树,它向我们揭示了在平衡的有向图中构造欧拉回路的简单方法。(请见习题 14 中的例子。)事实上,定理 D 允许我们来计算有向图中欧拉回路的精确个数;这一结果和在本小节建立的思想的其它许多重要结果出现在以下的习题中。

习 题

1.[*M20*] 证明:如果 V 和 V' 是一个有向图的顶点,而且如果从 V 到 V' 有一个有向的通路,则从 V 到 V' 存在简单的有向通路。

2.[*15*] 在 2.3.4.1 小节的(3)中列举的十个"基本回路"中哪些是在该小节的有向图(图 32)中的有向回路?

3.[*16*] 画出连通但不是有根的一个有向图的框图。

▶ 4.[*M20*] 拓扑排序的概念可以对任何有限的有向图 G 定义为顶点 $V_1V_2\cdots V_n$ 的线性安排,使得 G 的所有有向边 e 在这个排序下,$\mathrm{init}(e)$ 居于 $\mathrm{fin}(e)$ 之前。(请见 2.2.3 小节图 6 和 7。)并非所有有限的有向图都可拓扑地排序;那么哪些是可以的呢?(使用本小节的术语来给出答案。)

5.[*M16*] 设 G 是包含有向通路 $(e_1,\cdots,e_n)$,且 $\mathrm{fin}(e_n)=\mathrm{init}(e_1)$ 的一个有向图。使用在本小节定义的术语,给出 G 不是一个有向树的证明。

6.[*M21*] 真或假:有根的和不包含回路及有向回路的有向图是有向树。

▶ 7.[*M22*] 真或假:满足有向树定义中的性质 a)和 b),以及没有有向回路的有向图,是有向树。

8.[*HM40*] 试研究有向树自同构群的性质,即由顶点和有向边的所有排列 π 组成的群,对于它们有 $\mathrm{init}(e\pi)=\mathrm{init}(e)\pi$,$\mathrm{fin}(e\pi)=\mathrm{fin}(e)\pi$。

9.［*18*］通过对诸边赋予方向，画出对应于图 30 中的自由树的有向树，且以 G 作为根。

10.［*22*］通过使用表格 $P[1],\cdots,P[n]$，具有顶点 $V_1,\cdots,V_n$ 的有向树在计算机内可以表示如下：如果 V_j 是根，则 $P[j]=0$；否则 $P[j]=k$，如果有向边 $e[V_j]$ 是从 V_j 通向 V_k 的话。（于是 $P[1],\cdots,P[n]$ 和用于算法 2.3.3E 中的"父亲"表格相同。）

正文中说明了如何通过把任何要求的顶点选作根，把自由树转换成有向树。结果，有可能通过有根 R 的有向树开始，然后通过忽略有向边的方向而把这个有向树转换成自由树，最后赋予新的方向，得到以任何确定的顶点作为根的有向树。试设计实施这种转换的算法：由表格 $P[1],\cdots,P[n]$ 开始，表示有向树，而且给定整数 j，$1\leqslant j\leqslant n$，设计转换 P 表的算法，使得它仍表示相同的树，但以 V_j 作为根。

▶ 11.［*28*］使用习题 2.3.4.1-6 中的假定，但以 (a_k,b_k) 表示从 V_{a_k} 到 V_{b_k} 的有向边，试设计像在该算法一样不仅打印自由子树，而且打印出基本回路的算法。［提示：在习题 2.3.4.1-6 的解中给出的算法可以同上一题的算法组合起来。］

12.［*M10*］在这里定义的有向树和在 2.3 节开始处定义的有向树之间的对应中，树节点的度数等于对应顶点的入度还是出度？

▶ 13.［*M24*］试证明，如果 R 是(可能无穷的)有向图 G 的根，则 G 包含和 G 有同样顶点且以 R 为根的有向子树。（作为结果，总是有可能在像 2.3.4.1 小节的图 32 那样的流程图中选择自由子树使得它实际上是有向子树；如果在该框图中，我们选定 e''_{13}，e''_{19}，e_{20} 和 e_{17} 而不是 e'_{13}，e'_{19}，e_{23} 和 e_{15}，情况也将是这样。）

14.［*21*］设 G 是图 36 中所示的平衡有向图，且设 G' 是有顶点 V_0,V_1,V_2 和有向边 e_{01},e_{21} 的有向子树。以有向边 e_{12} 开始，找出满足定理 D 的条件的所有通路 P。

15.［*M20*］真或假：连通和平衡的有向图是强连通的。

▶ 16.［*M24*］在称为"钟"的流行的单人游戏中，通常的 52 张扑克牌面朝下地分成每叠四张的 13 叠；12 叠被安排成一个圆圈像钟的 12 小时那样，而第 13 叠放在中心处。通过打开在中心处的那叠牌顶部的牌开始这个单人游戏。如果该张牌的面值是 k，那就把该牌放在靠边第 k 叠牌处。(号码 1,2,…,13 等价于 A,2,…,10,J,Q,K。)通过打开第 k 叠处顶部的牌并把它放在靠近它的叠处，等等，直到我们达到这样一个时刻为止，即由于在指定的叠处已没有牌可翻了，因而我们再不能继续进行下去。（在这个游戏中游戏者毫无选择的余地，因为其规则完全确定要做什么。）当游戏结束时，如果所有的牌都已翻开，则游戏获胜。［参考：E. D. Cheney, *Patience* (Boston: Lee & Shepard, 1870), 62～65；按照 Whitmore Jones, *Games of Patience* (London: L. Upcott Gill, 1900)，第 7 章，在英国，这个游戏称做"旅行者的耐心"。］

试证明，这游戏将取胜，当且仅当下列有向图是有向树：顶点是 $V_1,V_2,\cdots,V_{13}$，有向边是 $e_1,e_2,\cdots,e_{12}$，其中 e_j 是从 V_j 通向 V_k 的，如果在开牌之后 k 是第 j 叠中底部的牌的话。

（特别是，如果对于 $j\neq 13$，第 j 叠底部的牌是"j"，容易明白，这游戏者肯定输。因为这张牌肯定没法再往下翻了。本题证明的结果给出玩这个游戏快得多的一种方法。）

17.［*M32*］假定扑克牌是随机地洗的，(在习题 16 中描述的)单人游戏获胜的概率是多少？当游戏结束时，恰好有 k 张牌未翻的概率是多少？

18.［*M30*］设 G 是有 $n+1$ 个顶点 $V_0,V_1,\cdots,V_n$ 和 m 个边 $e_1,e_2,\cdots,e_m$ 的图。通过为每个边赋予任意的方向而把 G 变成为有向图；然后以

$$a_{ij}=\begin{cases}+1,\text{如果 init}(e_i)=V_j\\ -1,\text{如果 fin}(e_i)=V_j\\ 0,\text{其它情况}\end{cases}$$

构造 $m\times(n+1)$的矩阵 A。设 A_0 是 A 删去第 0 列后的 $m\times n$ 矩阵。

a)若 $m=n$,证明:如果 G 不是自由树,则 A_0 的行列式等于 0;而如果 G 是自由树,则它等于 ±1。

b)证明:对于一般的 m, $A_0^TA_0$ 的行列式等于 G 的自由子树的个数(即从 m 个边选择 n 个使得得到的图是自由树的方式数)。[提示:使用 a)和习题 1.2.3-46 的结果。]

19.[*M31*](矩阵树定理) 设 G 是有 $n+1$ 个顶点 $V_0,V_1,\cdots,V_n$ 的有向图。设 A 是$(n+1)\times(n+1)$的矩阵,且

$$a_{ij}=\begin{cases}-k,\text{如果 } i\neq j \text{ 且从 } V_i \text{ 到 } V_j \text{ 有 } k \text{ 个有向边}\\ t,\text{如果 } i=j \text{ 且从 } j \text{ 到其它顶点有 } t \text{ 个有向边}\end{cases}$$

(由此得出,对于 $0\leqslant i\leqslant n$, $a_{i0}+a_{i1}+\cdots+a_{in}=0$。)令 A_0 是 A 去掉第 0 行和第 0 列后的矩阵。例如,如果 G 是图 36 的有向图,我们有

$$A=\begin{pmatrix}2&-2&0\\-1&3&-2\\-1&-1&2\end{pmatrix}\qquad A_0=\begin{pmatrix}3&-2\\-1&2\end{pmatrix}$$

a)证明:如果 $a_{00}=0$ 以及对于 $1\leqslant j\leqslant n$, $a_{jj}=1$,而且如果 G 不包含从一个顶点到它自身的有向边,则 A_0 的行列式 = [G 是以 V_0 为根的有向树]。

b)试证明:一般地说,A_0 的行列式等于根在 V_0 的 G 的有向子树的个数(即从 G 的有向边中选择 n 个使得得到的有向图是有向树且以 V_0 作为根的方式数)。[提示:对有向边数使用归纳法。]

20.[*M21*] 如果 G 是 $n+1$ 个顶点 $V_0,\cdots,V_n$ 上的一个无向图,并设 B 是对于 $1\leqslant i,j\leqslant n$ 如下定义的 $n\times n$ 矩阵:

$$b_{ij}=\begin{cases}t, & \text{如果 } i=j \text{ 且有 } t \text{ 条边触及 } V_j\\ -1, & \text{如果 } i\neq j \text{ 且 } V_i \text{ 和 } V_j \text{ 相邻}\\ 0, & \text{其它情况}\end{cases}$$

例如,如果 G 是前面图 29 的图,且$(V_0,V_1,V_2,V_3,V_4)=(A,B,C,D,E)$,可求得

$$B=\begin{pmatrix}3&0&-1&-1\\0&2&-1&0\\-1&-1&3&-1\\-1&0&-1&2\end{pmatrix}$$

试证明,G 的自由子树的个数是 B 的行列式。[提示:使用习题 18 或 19。]

21.[*HM38*](T. van Aardenne-Ehrenfest 和 N. G. de Bruijn)图 36 是不但平衡而且规则的有向图的例子。所谓规则,指每个顶点和所有其它顶点一样有相同的入度和出度。设 G 是有 n 个顶点 $V_0,V_1,\cdots,V_{n-1}$的规则的有向图,其中每个顶点的入度和出度都等于 m。(因此全部有 mn 个有向边。)设 G^* 是有 mn 个对应于 G 的有向边的顶点的有向图;设 G^* 对应于 G 中从 V_j 到 V_k 的有向边的一个顶点以 V_{jk}来表示。在 G^* 中的一个有向边从 V_{jk}通到 $V_{j'k'}$ 当且仅当 $k=j'$。例如,如果 G 是图 36 中的有向图,G^* 示于图 37 中。G 中的欧拉回路是 G^* 中的哈密顿回路,且反之亦然。

证明 G^* 中有向子树的个数是 $m^{(m-1)n}$乘以 G 的有向子树的个数。[提示:使用习题 19。]

▶ 22.[*M26*] 设 G 是具有顶点 $V_0,V_1,\cdots,V_n$ 且无孤立顶点的平衡的有向图。设 σ_j 是 V_j 的出度。证明 G 的欧拉回路的个数是

$$(\sigma_1+\sigma_2+\cdots+\sigma_n)T\prod_{j=1}^{n}(\sigma_j-1)!$$

其中 T 是以 V_1 为根的 G 的有向子树的个数。[注:作为 G 的有向边个数的因子($\sigma_1+\sigma_2+\cdots+\sigma_n$)可以省略,如果把欧拉回路($e_1,\cdots,e_m$)当做等于($e_k,\cdots,e_m,e_1,\cdots,e_{k-1}$)的话。]

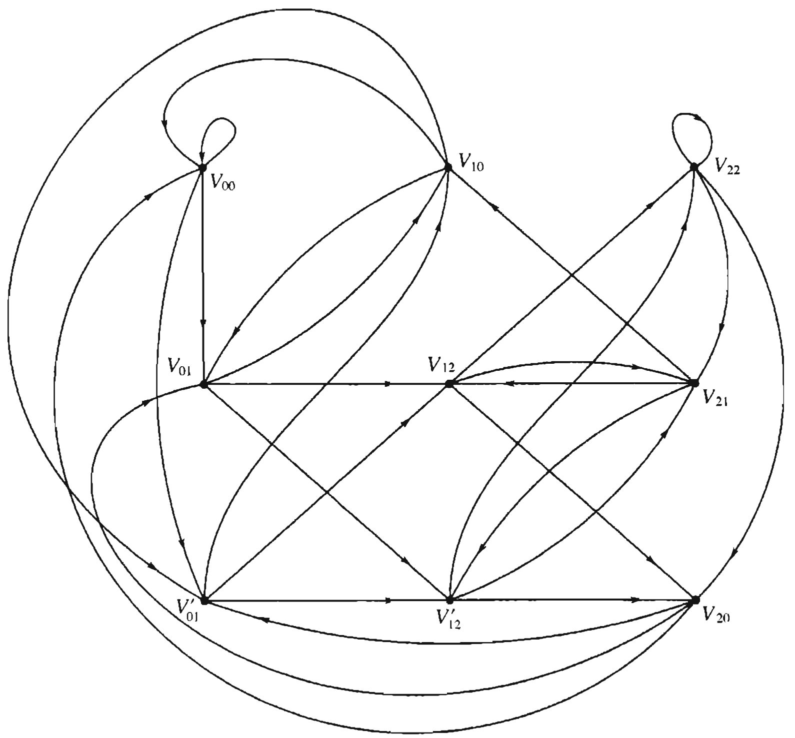


图 37 对应于图 36 的有向边的有向图(见习题 21)

▶ 23.[*M33*] (N.G.De Bruijn)对于小于 m 的每个非负整数序列 $x_1,\cdots,x_k$,设 $f(x_1,\cdots,x_k)$是小于 m 的非负整数。定义无穷序列如下:$X_1=X_2=\cdots=X_k=0$;$X_{n+k+1}=f(X_{n+k},\cdots,X_{n+1})$,当 $n\geqslant 0$ 时。对于 m^{m^k} 个可能的函数 f 来说,有多少个是其周期有极大长度的周期序列。[提示:对于所有 $0\leqslant x_j<m$ 的顶点($x_1,\cdots,x_{k-1}$),和对于从($x_1,x_2,\cdots,x_{k-1}$)到($x_2,\cdots,x_{k-1},x_k$)的有向边,构造一个有向图;应用习题 21 和 22。]

▶ 24.[*M20*] 设 G 是有有向边 $e_0,e_1,\cdots,e_m$的连通有向图。设 $E_0,E_1,\cdots,E_m$ 是对于 G 满足基尔霍夫定律的正整数的集合,即是,对于每个顶点 V,

$$\sum_{\mathrm{init}(e_j)=V}E_j=\sum_{\mathrm{fin}(e_j)=V}E_j$$

进一步假设 $E_0=1$。试证明在 G 中存在从 $\mathrm{fin}(e_0)$到 $\mathrm{init}(e_0)$的有向通路,使得对于 $1\leqslant j\leqslant m$,边 e_j 恰出现 E_j 次,而边 e_0 不出现。[提示:将定理 G 应用于合适的有向图上。]

25.[*26*] 试设计有向图的计算机表示,它可把树的右穿线二叉树表示加以推广。使用两个

链接字段 ALINK,BLINK 和两个一位的字段 ATAG,BTAG;并且设计一个表示,使得(i)对于有向图的每一个有向边(不是对于每个顶点)有一个节点;(ii) 如果有向图是以 R 为根的有向树,而且如果我们加上从 R 到新顶点 H 的一个有向边,则在 ALINK,BLINK,BTAG 分别和 2.3.2 小节中的 LLINK,RLINK,RTAG 相同的意义下,这个有向图的表示实质上和这个有向树的右穿线表示相同(只是在每个家庭中对儿子强加上某个顺序);以及(iii)在把 ALINK,ATAG 和 BLINK,BTAG 加以交换等价于有向图的所有有向边改变方向这一意义下,这个表示是对称的。

▶ 26.[*HM39*] (对随机算法的分析) 设 G 是顶点 $V_1, V_2, \cdots, V_n$ 上的有向图。假定 G 表示一个算法的流程图。其中 V_1 是开始顶点而 V_n 是停止顶点。(因此 V_n 是 G 的根。)假设对 G 的每个有向边赋以概率 $p(e)$,这些概率满足条件

$$0 \leqslant p(e) \leqslant 1;\ \sum_{\text{init}(e)=V_j} p(e) = 1, \text{对于 } 1 \leqslant j \leqslant n$$

考虑一条随机通路,启始于 V_1,接着选择概率为 $p(e)$的 G 的分支 e,直到达到 V_n 为止;在每步对分支的选择是和所有以前的选择无关的。

例如,考虑习题 2.3.4.1-7 的图,并分别分配概率 $1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 1, \frac{3}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}$ 到有向边 $e_1, e_2, \cdots, e_9$ 上。然后以概率 $1 \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{3}{4} \cdot 1 \cdot \frac{1}{4} = \frac{3}{128}$ 选定通路"开始—A—B—C—A—D—B—C—停止"。

为纪念俄罗斯数学家 Andrei A. Markov(马尔科夫),这样的随机通路称为马尔科夫链。马尔科夫首先对这类随机过程进行了广泛的研究。这种情况可作为某些算法的模型,尽管每种选择都必须同其它选择无关这样的要求是很强的假定。本题的目的是分析这类算法的计算时间。

通过考虑 $n \times n$ 矩阵 $A = (a_{ij})$ 可使这个分析容易进行,其中的 $a_{ij} = \sum p(e)$ 是对从 V_i 通到 V_j 的所有有向边 e 进行求和的。如果没有这样的有向边,则 $a_{ij} = 0$。上面考虑的例子的矩阵 A 是

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & \frac{3}{4} & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

容易得出,$(A^k)_{ij}$ 是在 k 步之后,由 V_i 开始的通路将处于 V_j 的概率。

试证明,对于所述类型的任意有向图 G 有以下事实:

a)矩阵$(I - A)$是非奇异的。[提示:证明没有非零向量 x,使 $xA^n = x$。]

b)顶点 V_j 在通路上出现的平均次数是

$$(I - A)^{-1}_{1j} = \text{cofactor}_{j1}(I - A)/\det(I - A), \quad \text{对于 } 1 \leqslant j \leqslant n$$

[于是在所考虑的例子中,我们发现,平均说来,顶点 A, B, C, D 分别被遍历 $\frac{13}{6}, \frac{7}{3}, \frac{7}{3}, \frac{5}{3}$ 次。]

c) V_j 出现在通路中的概率是

$$a_j = \text{cofactor}_{j1}(I - A)/\text{cofactor}_{jj}(I - A)$$

其次 $a_n = 1$,所以以概率 1,这个通路在有限步之内结束。

d)在 V_j 处开始的随机通路绝不会返回 V_j 的概率是 $b_j = \det(I - A)/\mathrm{cofactor}_{jj}(I - A)$。

e)对于 $k \geqslant 1, 1 \leqslant j \leqslant n$, V_j 在通路中出现恰好 k 次的概率是 $a_j(1 - b_j)^{k-1} b_j$。

27.[*M30*] (稳定状态)设 G 是顶点 $V_1, V_2, \cdots, V_n$ 上的有向图;其有向边已像在习题 26 中那样被赋予概率 $p(e)$。然而,没有开始和停止顶点,但假定 G 是强连通的;因此每个顶点 V_j 是根,而且我们假定概率 $p(e)$ 为正而且对所有 j 满足 $\sum_{\mathrm{init}(e) = V_j} p(e) = 1$。在习题 26 中所描述类型的随机过程称为具有"稳定的状态"$(x_1, \cdots, x_n)$,如果

$$x_j = \sum_{\substack{\mathrm{init}(e) = V_i \\ \mathrm{fin}(e) = V_j}} p(e) x_i, \qquad 1 \leqslant j \leqslant n$$

设 t_j 是对 G 的所有有向子树 T_j 的求和,这些子树是以 V_j 为根的,即乘积 $\prod_{e \in T_j} p(e)$ 之和。证明 $(t_1, \cdots, t_n)$ 是随机过程的稳定状态。

▶ 28.[*M35*] 考虑在这里所说明的 $m = 2$ 和 $n = 3$ 的 $(m+n) \times (m+n)$ 的行列式

$$\det\begin{pmatrix} a_{10} + a_{11} + a_{12} + a_{13} & 0 & a_{11} & a_{12} & a_{13} \\ 0 & a_{20} + a_{21} + a_{22} + a_{23} & a_{21} & a_{22} & a_{23} \\ b_{11} & b_{12} & b_{10} + b_{11} + b_{12} & 0 & 0 \\ b_{21} & b_{22} & 0 & b_{20} + b_{21} + b_{22} & 0 \\ b_{31} & b_{32} & 0 & 0 & b_{30} + b_{31} + b_{32} \end{pmatrix}$$

试证明,当把这个行列式展开成诸 a 和诸 b 的多项式时,每个非零项的系数为 $+1$。在这个展开式中有多少项出现?试给出同有向树相关联的规则,它精确地表征有哪些项出现。

*2.3.4.3　"无穷性引理"**　直到现在为止,我们主要地专注于仅有有限多个顶点(节点)的树,但是我们对于自由树和有向树给出的定义也适用于无穷的图。无穷有序树可用好几种方式定义;例如,我们可以把"杜威十进记号"的概念推广到数的无穷集合上,像在习题 2.3-14 中那样。甚至在计算机算法的研究中,有时也需要知道无穷树的性质——比如,通过反证法证明,某个树不是无穷的。D. König 以其充分的一般性,首先指出无穷树最基本的性质之一如下:

定理 K("无穷性引理")　*每个顶点具有有限的度的每个无穷有向树都有通向根的无穷通路,即有顶点为 $V_0, V_1, V_2, \cdots$ 的无穷序列,其中 V_0 是根且对于所有 $j \geqslant 0$,* $\mathrm{fin}(e[V_{j+1}]) = V_j$。

证明　我们以 V_0,即有向树的根开始定义通路。假定 $j \geqslant 0$,而且 V_j 已被选定有无穷多个后裔。由假设,V_j 的度数是有限的,所以 V_j 有有限多个儿子 $U_1, \cdots, U_n$。这些儿子中至少有一个必须具有无穷多个后裔,所以我们取 V_j 的这样的儿子作为 V_{j+1}。

现在 $V_0, V_1, V_2, \cdots$ 是通向根的一条无穷通路。▌

微积分的学生们可能会认识到,这里所用的论证实质上与用来证明经典的 Bolzano—Weierstrass 定理,即"有界无穷的实数集合具有聚积点"的论证一样。如同 König 所发现的那样,论述定理 K 的一种方法是:"如果人类永不灭绝,则现在活着的某

个人就会有不绝的后代。”

大多数人在第一次碰到定理 K 时会认为它是完全显然的，但在经过更多的思考和对进一步例子的考虑之后，他们才认识到关于此定理的某些深远的问题。尽管树的每个节点的度数是有限的，但我们并未假定，度是有界的(即对于所有顶点，小于某个数 N)，所以可能有度数越来越高的节点。因此至少这样一点是可以想像的，即尽管会有某些家庭，他们将延续百万代，还有的延续十亿代，等等，但是每个人的后代都将最终灭绝。事实上，H. W. Watson 首先发表了一个“证明”，即在不确定地进行下去的某些生物概率的定律之下，未来将有无穷多的人出生，但是每个家族都将以 1 的概率灭绝。他的论文[*J. Anthropological Inst. Gt. Britain and Ireland* **4** (1874), 138 ~ 144]包含了重要的和影响深远的定理，尽管小的疏忽引起他作出了这个论述。而且重要的是他没有发现他的结论在逻辑上是不一致的。

定理 K 的反命题可直接应用于计算机算法中：*如果我们有一种算法，可周期地把自己分成为有限多个子算法，而且如果每个子算法链最终终止，则这个算法本身也终止。*

换成另一种方式来叙述，假设我们有一个有限或无限的集合 S，使得 S 的每个元素是有限长度 $n \geq 0$ 的正整数序列$(x_1, x_2, \cdots, x_n)$。如果我们强加以下条件：

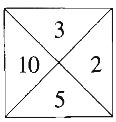
i)如果$(x_1, \cdots, x_n)$在 S 中，则对于 $0 \leq k \leq n$，$(x_1, \cdots, x_k)$也在 S 中。

ii)如果$(x_1, \cdots, x_n)$在 S 中，则只有有限多个 x_{n+1} 存在，对于它来说，$(x_1, \cdots, x_n, x_{n+1})$也在 S 中。

iii)不存在无穷的序列$(x_1, x_2, \cdots)$，其所有初始序列$(x_1, x_2, \cdots, x_n)$都在 S 中。

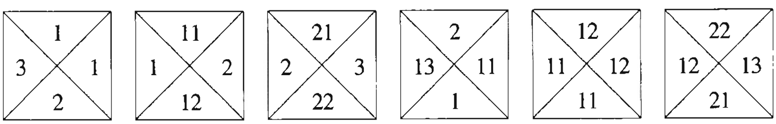
那么，S 实质上是以杜威十进记号确定的一个有向树，而且定理 K 告诉我们，S 是有限的。

定理 K 的效力最令人信服的例子之一，在同王浩所引进的有趣的一类平铺问题相联系时表现出来。四位一体型是分成为四个部分的正方形，其中每个部分都有特定的数字，例如



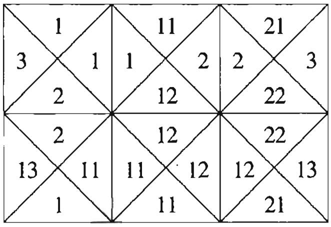
(1)

平铺平面的问题是使用四位一体型的有限集合，但每种四位一体型可以有无穷多个，看看如何把四位一体型放到无穷平面的每一正方形处(不许转动或翻转)，使得两个四位一体型仅当在接触处有相同数字时才可以相邻。例如，使用六个四位一体型



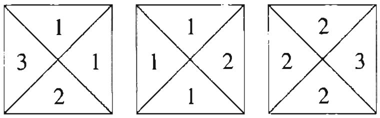
(2)

我们实质上仅能以一种方式，通过不断重复下列矩形



(3)

来平铺平面。读者可以容易地验证,用下列三种四位一体型



(4)

不可能平铺平面。

王浩的发现[*Scientific American* **213**,5(1965 年 11 月),98 ~ 106]如下:*如果有可能平铺平面的右上角四分之一,就有可能平铺整个平面*。这肯定是出乎意料之外的,因为平铺右上角四分之一象限的方法涉及沿着 x 轴和 y 轴的"边界",而且似乎对于如何平铺平面的左上四分之一象限并未提供任何提示(因为四位一体型不能旋转或翻转)。我们不能仅仅通过把右上角四分之一象限的解下移和左移就解决边界的问题,因为把解移动超过有限的数量并无意义。但是王浩给出证明如下:右上角四分之一象限解的存在意味着,对于所有的 n,存在一种方法来平铺 $2n \times 2n$ 的正方形。对于平铺每边有偶数个单元的正方形的问题的所有解的集合形成一个有向树,如果每个 $2n \times 2n$ 解 x 的儿子都是通过界定 x 可以得到的 $(2n+2) \times (2n+2)$ 的解的话。这个有向树的根是 0×0 的解。它的儿子是 2×2 的解,等等。每个节点仅有有限多个儿子,因为平铺平面问题假定,仅仅给定有限多个四位一体型;因此由无穷性引理,存在通向根的无穷的通路。这意味着,存在一种方法来平铺整个平面(尽管我们可能并不知道怎么找到它)!

关于平铺问题的更新进展,请参见 B. Grünbaum 和 G. C. Shephard 的佳作 *Tilings and Patterns* (Freeman, 1987),第 11 章。

习　题

1. [*M10*] 正文提及包含正整数的有限序列的集合 S,并且指出"这个集合实质上是有向树"。什么是这个有向树的根,什么是有向边?

2. [*20*] 证明如果允许四位一体型的转动,则总有可能平铺平面。

▶ 3. [*M23*] 如果当给定*无穷*的四位一体型的集合时有可能平铺平面的右上角四分之一象限,则是否总有可能平铺整个平面?

4. [*M25*](王浩) 六个四位一体型(2)导致平铺问题的环形解,即,其中某个矩形模式——即(3)——是在整个平面中被重复地平铺的。

不加证明地假定,每当有可能以有限的四位一体型的集合来平铺平面时,就存在有使用这些四位一体型的环形解。使用这个假定连同无穷性引理一起,设计一个这样的算法,给定任何四位

一体型的有限集合的特征描述，在有限步内确定是否存在一种方法通过这些四位一体型来平铺平面。

5. [*M40*] 试证明：使用以下 92 个四位一体型有可能平铺平面，但是没有在习题 4 意义下的环形解。

为简化这 92 种类型的特征描述，让我们首先介绍一些记号。定义下列的"基本代码"：

$\alpha=(1,2,1,2)$	$\beta=(3,4,2,1)$	$\gamma=(2,1,3,4)$	$\delta=(4,3,4,3)$
$a=(Q,D,P,R)$	$b=(\ ,\ ,L,P)$	$c=(U,Q,T,S)$	$d=(\ ,\ ,S,T)$
$N=(Y,\ ,X,\ )$	$J=(D,U,\ ,X)$	$K=(\ ,Y,R,L)$	$B=(\ ,\ ,\ ,\ )$
$R=(\ ,\ ,R,R)$	$L=(\ ,\ ,L,L)$	$P=(\ ,\ ,P,P)$	$S=(\ ,\ ,S,S)$
	$T=(\ ,\ ,T,T)$	$X=(\ ,\ ,X,X)$	
$Y=(Y,Y,\ ,\ )$	$U=(U,U,\ ,\ )$	$D=(D,D,\ ,\ )$	$Q=(Q,Q,\ ,\ )$

四位一体型现在是

$\alpha\{a,b,c,d\}$ [4 个类型]

$\beta\{Y\{B,U,Q\},\{P,T\},\{B,U,D,Q\},\{P,S,T\},K\{B,U,Q\}\}$ [21 个类型]

$\gamma\{\{\{X,B\},\{L,P,S,T\},R\{\{B,Q\},J\{L,P,S,T\}\}$ [22 个类型]

$\delta\{X\{L,P,S,T\}\{B,Q\},Y\{B,U,Q\}\{P,T\},N\{a,b,c,d\}$
$J\{L,P,S,T\},K\{B,U,Q\},\{R,L,P,S,T\}\{B,U,D,Q\}\}$ [45 个类型]

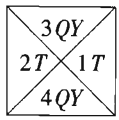
这些缩写意味着基本代码被逐个分量地放在一起并且在每个分量中按字母顺序排序；于是

$$\beta Y\{B,U,Q\}\{P,T\}$$

代表 6 种类型 $\beta YBP,\beta YUP,\beta YQP,\beta YBT,\beta YUT,\beta YQT$。在把相应分量相乘和排序之后，类型 βYQT 是

$$(3,4,2,1)(Y,Y,\ ,)(Q,Q,\ ,)(\ ,\ ,T,T)=(3QY,4QY,2T,1T)$$

这是有意地来对应于右边所示的四位一体型，其中我们在类型的四个四分之一中，使用符号串而不是数字。如果两个四位一体型在它们相接触处有相同的符号串，它们就可以彼此紧挨着放在一起。



β 四位一体型是如同上面给出的那样在其特征描述中有 β 的类型。为开始本题的求解，注意任何 β 四位一体必须在它的左边和右边有 α 四位一体，以及在它上面和下面有 δ 四位一体型。αa 四位一体型必须有 βKB 或 βKU 或 βKQ 在它的右边，因而必须成为 αb 四位一体型，等等。

（这个构造是由 Robert Berger 给出的类似构造的简化版本。他继续证明，如无不正确的假定，习题 4 中的一般问题，不可能被求解。请见*Memoires Ameri. Math. Soc.* **66** (1966)。）

▶6. [*M23*] (Otto Schreier) 在一篇有名的论文[*Nieuw Archief voor Wiskunde* (2) **15** (1927), 212～216]中，B. L. van Waerden 证明了下列定理：

如果 k 和 m 是正整数，而且如果我们有 k 个正整数的集合 $S_1,\cdots,S_k$ 且每个正整数至少被包含在这些集合之一，则至少有集合 S_j 之一包含长度为 m 的算术级数。

（后一个命题意味着，存在整数 a 和 $\delta>0$ 使得 $a+\delta,a+2\delta,\cdots,a+m\delta$ 全都在 S_j 中。）如果可能，利用这一结果和无穷性引理来证明下列更强的命题：

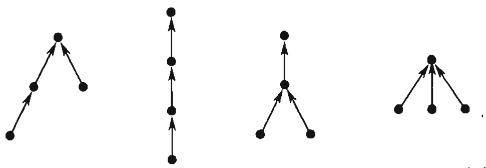
如果 k 和 m 是正整数，则存在一个数 N 使得如果我们有 k 个整数集合且 1 和 N 之间的每一个整数至少被包括在这些集合之一中，则集合 S_j 中至少有一个包含长度为 m 的算术级数。

▶7.[*M30*] 如果可能,利用习题 6 的 van der Waerden 定理和无穷性引理证明下列更强的命题:

如果 k 是正整数,而且如果我们有 k 个整数集合 $S_1,\cdots,S_k$ 且每个正整数被包括在这些集合的至少一个之中,则集合 S_j 中至少有一个包含无穷长的算术级数。

▶8.[*M39*](J.B.Kruskal) 如果 T 和 T' 都是(有限的,有序的)树,令记号 $T\subseteq T'$ 如同在习题 2.3.2-22 中一样,表示 T 可被嵌入到 T' 中。证明如果 $T_1,T_2,T_3,\cdots$ 是树的任何无穷序列,则存在整数 $j<k$ 使得 $T_j\subseteq T_k$。(换句话说,不可能来构造这样一个树的无穷序列,其中没有一个树包含这个序列中任何较早的树。这个事实可用来证明某些算法必然终止。)

***2.3.4.4　树的枚举**　树的数学理论对于算法分析最有教益的某些应用,是同计算有多少种各种不同类型的树的数目的公式相关联的。例如,如果我们想要知道有四个不可区分的顶点,可以构造出多少个不同类型的有向树,我们发现,恰好有 4 种可能性:


(1)

作为第一个枚举问题,让我们确定具有 n 个顶点的在结构上不同的有向树的个数 a_n。显然,$a_1=1$。如果 $n>1$,则这个树有一个根和各种子树;假设有 j_1 个 1 个顶点的子树,有 j_2 个 2 个顶点的子树,等等。于是我们可用

$$\binom{a_k+j_k-1}{j_k}$$

种方式选择 a_k 个可能的 k 顶点树中的 j_k 个,因为允许重复(习题 1.2.6-60),因此我们看到当 n>1 时,

$$a_n=\sum_{j_1+2j_2+\cdots=n-1}\binom{a_1+j_1-1}{j_1}\cdots\binom{a_{n-1}+j_{n-1}-1}{j_{n-1}} \tag{2}$$

如果我们考虑生成函数 $A(z)=\sum_n a_n z^n$ 且 $a_0=0$,我们发现恒等式

$$\frac{1}{(1-z^r)^a}=\sum_j\binom{a+j-1}{j}z^{rj}$$

连同(2)意味着

$$A(z)=\frac{z}{(1-z)^{a_1}(1-z^2)^{a_2}(1-z^3)^{a_3}\cdots} \tag{3}$$

这对于 $A(z)$ 不是特别漂亮的形式,因为它涉及无穷乘积而且系数 $a_1,a_2,\cdots$ 出现于等式的右边。习题 1 中给出表示 $A(z)$ 的较为优雅的方式;它导致了计算 a_n 相当有效的公式(见习题 2),而且事实上,对于很大的 n,它也可用来推导 a_n 的渐进特性(见习题 4)。我们求得

$$A(z)=z+z^2+2z^3+4z^4+9z^5+20z^6+48z^7+115z^8+$$
$$286z^9+719z^{10}+1842z^{11}+\cdots \tag{4}$$

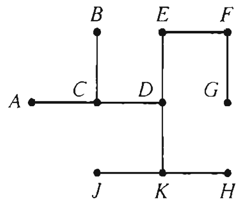
既然实际上我们已经找到有向树的个数,则确定有 n 个顶点的结构上不同的自由树的个数是十分有趣的。对于四个顶点,恰好有两个不同的树,就是



(5)

因为(1)的前两个和后两个在去掉方向时变成相同的了。

我们已经看到,有可能选择自由树的任何顶点 X 并且以惟一的方式来为边指定方向,使得它变成以 X 作为根的有向树。对于给定的顶点 X,一旦这已完成,假设有以 X 为根的 k 个子树,且在这些分别的子树中有顶点数 $s_1, s_2, \cdots, s_k$。显然,k 是同 X 接触的有向边的个数,而且 $s_1 + s_2 + \cdots + s_k = n - 1$。则在这些情况下我们说 X 的权是 $\max(s_1, s_2, \cdots, s_k)$。因此在树



(6)

中,顶点 D 有权 3(从 D 引出的每个子树有剩下的九个顶点中的三个),而顶点 E 有权 $\max(7,2) = 7$,具有极小权的顶点称为自由树的*形心*。

令 X 和 $s_1, s_2, \cdots, s_k$ 如上,并令 $Y_1, Y_2, \cdots, Y_k$ 是从 X 发散出的子树的根。如果 Y 是 Y_1 子树中的任一节点,则它的权必定至少为 $n - s_1 = 1 + s_2 + \cdots + s_k$,因为当 Y 是假定的根时,在其包含 X 的子树中有 $n - s_1$ 个顶点。因此如果 Y 是形心,我们有

$$\text{权}(X) = \max(s_1, s_2, \cdots, s_k) \geqslant \text{权}(Y) \geqslant 1 + s_1 + s_2 + \cdots + s_k$$

而且仅当 $s_1 > s_2 + \cdots + s_k$ 时这才可能。如果在这个讨论中我们以 Y_j 代替 Y_1,可以导出类似的结果。所以*在一个顶点的子树中至多有一个子树可以包含形心*。

这是一个很强的条件,因为它意味着*在一个自由树中至多有两个形心,而且如果有两个形心存在,它们是相邻的*。(见习题 9。)

反之,如果 $s_1 > s_2 + \cdots + s_k$,则在 Y_1 子树处有一个形心,因为

$$\text{权}(Y_1) \leqslant \max(s_1 - 1, 1 + s_2 + \cdots + s_k) \leqslant s_1 = \text{权}(X)$$

而且在 $Y_2, \cdots, Y_k$ 子树处所有节点的权至少是 $s_1 + 1$。我们已经证明*顶点 X 是一个自由树的惟一形心,当且仅当对于 $1 \leqslant j \leqslant k$,*

$$s_j \leqslant s_1 + s_2 + \cdots + s_k - s_j \tag{7}$$

因此有 n 个顶点且仅有一个形心的自由树的数目,是具有 n 个顶点的有向树的数目减去违反条件(7)的这样的有向树的数目;后者实质上由有 s_j 个顶点的有向树和有 $n - s_j \leqslant s_j$ 个顶点的另一个有向树组成。因此具有一个形心的自由树的个数成为

$$a_n - a_1 a_{n-1} - a_2 a_{n-2} - \cdots - a_{\lfloor n/2 \rfloor} a_{\lceil n/2 \rceil} \tag{8}$$

具有两个形心的一个自由树有偶数个顶点，而且每一个形心的权是 $n/2$（见习题 10）。所以如果 $n=2m$，则二形心自由树的个数是允许重复地由 a_m 个事物中选择 2 个的选择数，即

$$\binom{a_m+1}{2}$$

因此，为了获得自由树的总数，我们在 n 为偶数时把 $\frac{1}{2}a_{n/2}(a_{n/2}+1)$ 加到(8)中。等式(8)的形式提示一个简单的生成函数，而且确实，我们毫无困难地发现，*对于结构上不同的自由树的个数的生成函数是*

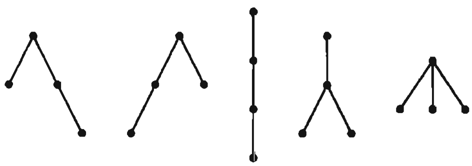
$$F(z) = A(z) - \frac{1}{2}A(z)^2 + \frac{1}{2}A(z^2) =$$

$$z + z^2 + z^3 + 2z^4 + 3z^5 + 6z^6 +$$

$$11z^7 + 23z^8 + 47z^9 + 106z^{10} + 235z^{11} + \cdots \tag{9}$$

$F(z)$和 $A(z)$之间的这个简单关系主要归功于 C. Jordan，他在 1869 年考虑了这个问题。

现在让我们转到枚举有序树的问题，对于计算机程序设计的算法它是我们关心的主要问题。对于四个顶点来说有五种结构上不同的有序树



(10)

前两种作为有向树是相同的，所以在上面的(1)中只有其中之一出现。

在考察不同的有序树结构之前，让我们首先考虑二叉树的情况，因为这更接近于实际的计算机表示而且它也更易于研究。设 b_n 是有 n 个节点的不同二叉树的数目。由二叉树的定义，很显然 $b_0=1$，而且对于 $n>0$，可能的数目是把具有 k 个节点的二叉树放到根的左边和把具有 $n-1-k$ 个节点的另一个二叉树放到右边的方式数，所以

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_{n-1} b_0, \quad n \geqslant 1 \tag{11}$$

由这个关系，显然，生成函数

$$B(z) = b_0 + b_1 z + b_2 z^2 + \cdots$$

满足方程

$$zB(z)^2 = B(z) - 1 \tag{12}$$

求解这个二次方程并且使用 $B(0)=1$ 这一事实，我们得到

$$B(z)=\frac{1}{2z}(1-\sqrt{1-4z})=\frac{1}{2z}\left(1-\sum_{k\geq 0}\binom{\frac{1}{2}}{k}(-4z)^k\right)=$$

$$2\sum_{n\geq 0}\binom{\frac{1}{2}}{n+1}(-4z)^n=\sum_{n\geq 0}\binom{-\frac{1}{2}}{n}\frac{(-4z)^n}{n+1}=$$

$$\sum_{n\geq 0}\binom{2n}{n}\frac{z^n}{n+1}=$$

$$1+z+2z^2+5z^3+14z^4+42z^5+132z^6+429z^7+$$

$$+1430z^8+4862z^9+16796z^{10}+\cdots \tag{13}$$

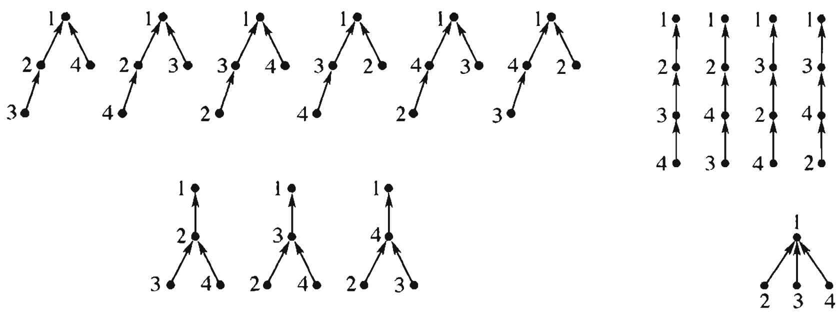
(见习题 1.2.6-47。)因此所求答案为

$$b_n=\frac{1}{n+1}\binom{2n}{n} \tag{14}$$

由斯特林公式,这可渐近地表示为 $4^n/n\sqrt{\pi n}+O(4^n n^{-5/2})$。在习题 11 和 32 中可找到等式(14)的一些重要推广。

回到我们关于有 n 个节点的有序树的问题,可以看出这实质上和二叉树的个数的问题是相同的,因为在二叉树和森林之间存在自然的对应,而且一棵树减去根就成为森林。因此*具有 n 个顶点的(有序)树的个数是 b_{n-1},即具有 $n-1$ 个顶点的二叉树的个数*。

上面所实现的枚举假定该顶点是不可区分的。如果我们在(1)中把顶点标以 1,2,3,4 并且坚持 1 作为根,那现在我们得到 16 个不同的有向树



(15)

对于标号树的枚举问题显然和上面解决的问题十分不同。在这种情况下它可以重新表达如下:"考虑画三条线,从顶点 2,3 和 4 之一指向另一个顶点,从每一个顶点发散的线有三种选择,所以全部共有 $3^3=27$ 种可能性。在这 27 种方式当中有多少将产生以 1 作为根的有向树呢?"如同我们已经见到的那样,答案是 16。对于同一个问题的类似的重新阐述,但这次是对于 n 个顶点的情况,如下:"设 $f(x)$ 是一个整数值函数,使得 $f(1)=$

1,而且对于所有整数 $1 \leqslant x \leqslant n$,有 $1 \leqslant f(x) \leqslant n$。如果对于所有的 x,$f^{[n]}(x)$ 即 $f(f(\cdots(f(x))\cdots))$迭代 n 次,等于 1,我们就称 f 为一个树的映射。问共有多少树的映射?"例如,这个问题在同随机数生成相关联时出现。相当令人惊讶的是,我们将发现,平均说来,每 n 个这样的函数 f 中恰好有一个是树的映射。

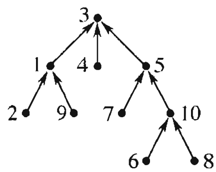
利用前一小节中研究过的计算图的子树的通用公式,可很容易地推出这一枚举问题的解(参见习题 12)。但是对于求解这个问题还有一个提供更多知识的方法,它为我们提供新的和紧凑的方式来表示有向树结构。

假设给了我们具有顶点$\{1,2,\cdots,n\}$和有 $n-1$ 个有向边的有向树,其中有向边是对于除根之外所有的 j,从 j 导向$f(j)$的。至少有一个终端(叶)顶点;设 V_1 是叶的最小号码。如果 $n>1$,写下 $f(V_1)$并从树中删除 V_1 和有向边 $V_1 \to f(V_1)$;然后设 V_2 是在得到的树中其顶点是终端的最小号码。如果 $n>2$,写下 $f(V_2)$并从树删除 V_2 和有向边 $V_2 \to f(V_2)$。如此继续进行直到除根之外所有顶点都已被删除为止。得到的 $n-1$ 个数的序列是

$$f(V_1),f(V_2),\cdots,f(V_{n-1}),\quad 1 \leqslant f(V_j) \leqslant n \tag{16}$$

这称做原来的有向树的*典型表示*。

例如有 10 个顶点的有向树



(17)

其典型表示为 1,3,10,5,10,1,3,5,3。

这里的重要一点是我们可以逆转这一过程,而且从 $n-1$ 个数的任何序列(16)转回到产生它的有向树。因为如果我们有 1 和 n 之间的树的任何序列$x_1,x_2,\cdots,x_{n-1}$,设 V_1 是在序列 $x_1,x_2,\cdots,x_{n-1}$中不出现的最小的数;然后设 V_2 是在序列 $x_2,\cdots,x_{n-1}$中不出现且$\neq V_1$ 的最小的数,等等。在以这种方式得到整数$\{1,2,\cdots,n\}$的一个排列 $V_1,V_2,\cdots,V_n$ 之后,对于$1 \leqslant j<n$,画出从顶点 V_j 到顶点 x_j 的有向边。这就给出没有有向回路的有向图构造,而且由习题 2.3.4.2-7,这是有向树。显然,序列 $x_1,x_2,\cdots,x_{n-1}$对于这一有向树与序列(16)是相同的。

由于这个过程是可逆的,我们就得到数$\{1,2,\cdots,n\}$的 $n-1$ 元组和这些顶点的有向树之间的一一对应。因此,*有 n 个标号顶点的不同有向树共有n^{n-1}个*。如果我们指定一个顶点作为根,显然在一个顶点和另外的顶点之间没有区别,因此在$\{1,2,\cdots,n\}$上有给定的根的不同有向树共有 n^{n-2}个,这就说明了(15)中的 $16=4^{4-2}$个树。从这个信息容易确定具有标号顶点的*自由*树的个数(见习题 22)。一旦我们知道不涉及标号的问题的答案时,也能容易地确定带有标号的*有序*树的个数(见习题 23)。所以,我们实质上已经解决了带有标号的顶点和不带有标号顶点的三种基本种类的树的枚举问题。

看看如果我们真的把通常的生成函数的方法应用到枚举带标号的有向树问题,那

会发生什么情况是有趣的。为此目的,我们大概将发现最容易的是考虑量 $r(n,q)$,即有 n 个顶点,且无有向回路,以及从 q 个指定的顶点的每一个都发出一条有向边的带标号有向图的个数。带有特定的根的带标号有向树的个数因此是 $r(n,n-1)$。在这个记号下,通过简单的计算论证,我们发现,对于任何固定的整数 m,

$$r(n,q)=\sum_k\binom{q}{k}r(m+k,k)r(n-m-k,q-k)\quad \text{如果 } 0\leqslant m\leqslant n-q \tag{18}$$

$$r(n,q)=\sum_k\binom{q}{k}r(n-1,q-k)\quad \text{如果 } q=n-1 \tag{19}$$

如果我们把未指定的顶点分成 A 和 B 两个组,且 m 个顶点在 A 中和 $n-q-m$ 个顶点在 B 中,就可得到这些关系的第一个;然后把 q 个指定的顶点分划成开始导向 A 的通路的 k 个顶点和开始导向 B 的通路的 $q-k$ 个顶点。通过考虑其中根的度为 k 的有向树,便得到关系(19)。

这些关系的形式指出,我们可以用生成函数

$$G_m(z)=r(m,0)+r(m+1,1)z+\frac{r(m+2,2)z^2}{2!}+\cdots=\sum_k\frac{r(k+m,k)z^k}{k!}$$

有利地工作。根据以上论述,等式(18)指出,$G_{n-q}(z)=G_m(z)G_{n-q-m}(z)$,而且因此由对 m 的归纳法,我们求得 $G_m(z)=G_1(z)^m$。现在从等式(19)得到

$$G_1(z)=\sum_{n\geqslant1}\frac{r(n,n-1)z^{n-1}}{(n-1)!}=\sum_{k\geqslant0}\sum_{n\geqslant1}\frac{r(n-1,n-1-k)z^{n-1}}{k!(n-1-k)!}=$$

$$=\sum_{k\geqslant0}\frac{z^k}{k!}G_k(z)=\sum_{k\geqslant0}\frac{(zG_1(z))^k}{k!}=e^{zG_1(z)}$$

换句话说,置 $G_1(z)=w$,我们问题的解来自于超越方程

$$w=e^{zw} \tag{20}$$

的解的系数。

这个方程可以通过使用拉格朗日的反演公式来求解:$z=\zeta/f(\zeta)$意味着

$$\zeta=\sum_{n\geqslant1}\frac{z^n}{n!}g_n^{(n-1)}(0) \tag{21}$$

其中 $g_n(\zeta)=f(\zeta)^n$,当 f 在原点的邻域中解析,而且 $f(0)\neq0$ 时(见习题4.7-16)。在这种情况下,我们可以置 $\zeta=zw$,$f(\zeta)=e^\zeta$,而且我们可以推出同上边得到的答案相一致的解

$$w=\sum_{n\geqslant0}\frac{(n+1)^{n-1}}{n!}z^n \tag{22}$$

G.N.Raney 已经证明,可以以一种重要的方式推广这个方法,得到更一般得多的方程

$$w = y_1 e^{z_1 w} + y_2 e^{z_2 w} + \cdots + y_s e^{z_s w}$$

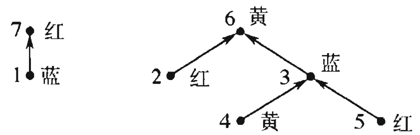
的解的显式幂级数，并且借助于 $y_1,\cdots,y_s$ 和 $z_1,\cdots,z_s$ 的幂级数来对 w 求解。对于这个推广，让我们考虑 s 维整数向量

$$\boldsymbol{n} = (n_1, n_2, \cdots, n_s)$$

而且为了方便起见我们写

$$\sum \boldsymbol{n} = n_1 + n_2 + \cdots + n_s$$

假设我们有 s 种颜色 $C_1, C_2, \cdots, C_s$，并且考虑对每个顶点都指定了一个颜色的有向图；例如



(23)

设 $r(\boldsymbol{n},\boldsymbol{q})$ 是画有向边以及对顶点 $\{1,2,\cdots,n\}$ 指定颜色的方式数，使得

i) 对于 $1 \leqslant i \leqslant s$，恰好有颜色 C_i 的 n_i 个顶点（因此 $n = \sum \boldsymbol{n}$）；

ii) 有 q 个有向边，从每个顶点 $\{1,2,\cdots,q\}$ 引出一个；

iii) 对于 $1 \leqslant i \leqslant s$，恰好有 q_i 个有向边导向颜色 C_i 的顶点（因此 $q = \sum \boldsymbol{q}$）；

iv) 没有有向回路（因此 $q < n$，除非 $q = n = 0$ 时）。

让我们把这称做一个 $(\boldsymbol{n},\boldsymbol{q})$ 构造。

例如，如果 $C_1 =$ 红，$C_2 =$ 黄，以及 $C_3 =$ 蓝，则 (23) 表明一个 $((3,2,2),(1,2,2))$ 构造。当仅有一种颜色时，我们有已经求解了的有向树问题。Raney 的想法是把这个一维的构造推广到 s 维中。

设 $\boldsymbol{n}$ 和 $\boldsymbol{q}$ 是非负整数的固定 s 位置的向量，且设 $n = \sum \boldsymbol{n}$，$q = \sum \boldsymbol{q}$，对于每个 $(\boldsymbol{n},\boldsymbol{q})$ 构造和每个数 k，$1 \leqslant k \leqslant n$，我们将定义由四件事组成的典型表示：

a) 数 t，且 $q < t \leqslant n$；

b) n 个颜色的序列，且颜色 C_i 有 n_i 个；

c) q 个颜色的序列，且颜色 C_i 有 q_i 个；

d) 对于 $1 \leqslant i \leqslant s$，集合 $\{1,2,\cdots,n_i\}$ 中的 q_i 个元素的序列。

典型表示定义如下：首先以有向树的典型表示的顺序 $V_1, V_2, \cdots, V_q$ 列出顶点 $\{1,2,\cdots,q\}$（如上面给出的那样），而后在顶点 V_j 下面写出在从 V_j 导出的有向边上的顶点的号码 $f(V_j)$。令 $t = f(V_q)$；并设颜色的序列 c) 仍分别是顶点 $f(V_1),\cdots,f(V_q)$ 的颜色。令颜色的序列 b) 分别是 $k, k+1, \cdots, n, 1, \cdots, k-1$ 顶点的颜色。最后，令在 d) 中的第 i 个序列是 $x_{i1}, x_{i2}, \cdots, x_{iq_i}$，其中，如果序列 $f(V_1),\cdots,f(V_q)$ 的第 j 个着了 C_i 颜色的元素是序列 $k, k+1, \cdots, n, 1, \cdots, k-1$ 的第 m 个着了 C_i 颜色的元素，则 $x_{ij} = m$。

例如，考虑构造 (23) 并设 $k = 3$。通过在它们之下列出 $V_1, \cdots, V_5$ 和 $f(V_1), \cdots, f(V_5)$，如下来开始：

1 2 4 5 3

7 6 3 3 6

因此 $t=6$,而且序列 c)分别表示颜色 7,6,3,3,6,即红,黄,蓝,蓝,黄。序列 b)分别表示颜色 3,4,5,6,7,1,2,即蓝,黄,红,黄,红,蓝,红。最后,为了得到 d)中的序列,如下进行:

颜色	在 3,4,5,6,7,1,2 中这个颜色的元素	在 7,6,3,3,6 中这个颜色的元素	通过列 2 来对列 3 进行编码
红	5,7,2	7	2
黄	4,6	6,6	2,2
蓝	3,1	3,3	1,1

因此 d)序列是 2;2,2;以及 1,1。

由典型表示,我们可发现原始的$(\boldsymbol{n},\boldsymbol{q})$构造以及数 k 如下:从 a)和 c)我们知道顶点 t 的颜色。对于这个颜色,d)序列的最后元素同 b)一起告诉我们,在序列 $k,\cdots,n,1,\cdots,k-1$ 中 t的位置;因此我们知道 k 和所有顶点的颜色。然后 d)中的子序列连同 b)和 c)一起确定 $f(V_1),f(V_2),\cdots,f(V_q)$,而且最后如同我们对于有向树所做的那样,通过确定 $V_1,\cdots,V_q$ 的位置重新构造出有向图来。

这个典型表示的可逆性允许我们计算可能的$(\boldsymbol{n},\boldsymbol{q})$构造的个数,因为对于 a)有 $n-q$ 个选择,对 b)有多项式系数

$$\binom{n}{n_1,\cdots,n_s}$$

这么多选择,以及对 c)有

$$\binom{q}{q_1,\cdots,q_s}$$

种选择,以及对 d)有 $n_1^{q_1}n_2^{q_2}\cdots n_s^{q_s}$ 种选择。再除以对于 k 的 n 种选择,我们有一般的结果:

$$r(\boldsymbol{n},\boldsymbol{q})=\frac{n-q}{n}\frac{n!}{n_1!\cdots n_s!}\frac{q!}{q_1!\cdots q_s!}n_1^{q_1}n_2^{q_2}\cdots n_s^{q_s} \tag{24}$$

其次,我们能导出等式(18)和(19)的类似式

$$r(\boldsymbol{n},\boldsymbol{q})=\sum_{\substack{\boldsymbol{k},\boldsymbol{t}\\ \sum(\boldsymbol{t}-\boldsymbol{k})=m}}\binom{\sum\boldsymbol{q}}{\sum\boldsymbol{k}}r(\boldsymbol{t},\boldsymbol{k})r(\boldsymbol{n}-\boldsymbol{t},\boldsymbol{q}-\boldsymbol{k}),\quad \text{如果 } 0\leqslant m\leqslant\sum(\boldsymbol{n}-\boldsymbol{q}) \tag{25}$$

其中约定 $r(\mathbf{0},\mathbf{0})=1$,以及 $r(\boldsymbol{n},\boldsymbol{q})=0$,如果任何 n_i 或 q_i 为负的话或如果 $q>n$ 的话;以及

$$r(\boldsymbol{n},\boldsymbol{q}) = \sum_{i=1}^{s}\sum_{k}\binom{\sum \boldsymbol{q}}{k} r(\boldsymbol{n}-\boldsymbol{e}_i,\boldsymbol{q}-k\boldsymbol{e}_i),\text{如果}\sum \boldsymbol{n} = 1 + \sum \boldsymbol{q} \qquad (26)$$

其中 $\boldsymbol{e}_i$ 是在位置 i 的值为 1 而在其它位置为 0 的向量。关系(25)是基于把顶点 $\{q+1,\cdots,n\}$ 分成分别有 m 个元素和有 $n-q-m$ 个元素的两部分;第二个关系通过删去惟一的根并考虑剩余的结构而导出。我们现在得到下列结果:

定理 R(George N. Raney, *Canadian J. Math.* **16** (1964), 755 ~ 762) 令

$$w = \sum_{\substack{\boldsymbol{n},\boldsymbol{q}\\ \sum(\boldsymbol{n}-\boldsymbol{q})=1}} \frac{r(\boldsymbol{n},\boldsymbol{q})}{(\sum \boldsymbol{q})!} y_1^{n_1}\cdots y_s^{n_s} z_1^{q_1}\cdots z_s^{q_s} \qquad (27)$$

其中 $r(\boldsymbol{n},\boldsymbol{q})$ 由(24)定义,而且其中 $\boldsymbol{n},\boldsymbol{q}$ 是 s 维整数向量。于是 w 满足恒等式

$$w = y_1 e^{z_1 w} + y_2 e^{z_2 w} + \cdots + y_s e^{z_s w} \qquad (28)$$

证明　由(25)和对 m 的归纳法,我们求得

$$w^m = \sum_{\substack{\boldsymbol{n},\boldsymbol{q}\\ \sum(\boldsymbol{n}-\boldsymbol{q})=m}} \frac{r(\boldsymbol{n},\boldsymbol{q})}{(\sum \boldsymbol{q})!} y_1^{n_1}\cdots y_s^{n_s} z_1^{q_1}\cdots z_s^{q_s} \qquad (29)$$

现在由(26),

$$w = \sum_{i=1}^{s}\sum_{k}\sum_{\substack{\boldsymbol{n},\boldsymbol{q}\\ \sum(\boldsymbol{n},\boldsymbol{q})=1}} \frac{r(\boldsymbol{n}-\boldsymbol{e}_i,\boldsymbol{q}-k\boldsymbol{e}_i)}{k!(\sum \boldsymbol{q}-k)!} y_1^{n_1}\cdots y_s^{n_s} z_1^{q_1}\cdots z_s^{q_s} =$$

$$\sum_{i=1}^{s}\sum_{k}\frac{1}{k!} y_i z_i^k \sum_{\substack{\boldsymbol{n},\boldsymbol{q}\\ \sum(\boldsymbol{n}-\boldsymbol{q})=k}} \frac{r(\boldsymbol{n},\boldsymbol{q})}{(\sum \boldsymbol{q})!} y_1^{n_1}\cdots y_s^{n_s} z_1^{q_1}\cdots z_s^{q_s} =$$

$$\sum_{i=1}^{s}\sum_{k}\frac{1}{k!} y_i z_i^k w^k \quad \blacksquare$$

在(27)和(28)中的 $s=1$ 和 $z_1=1$ 的特殊情况,在应用中特别重要,因此这种情况就成为熟知的"树函数"

$$T(y) = \sum_{n\geq 1}\frac{n^{n-1}}{n!} y^n = y e^{T(y)} \qquad (30)$$

关于这个函数的历史及值得注意的一些性质的讨论,请见 Corless, Gonnet, Hare, Jeffrey 及 Knuth, *Advances in Computational Math.* **5** (1996), 329 ~ 359。

基于对生成函数的巧妙操作,对于树的枚举的综述,已由 I. J. Good 给出 [*Proc. Cambridge Philos. Soc.* **61** (1965), 499 ~ 517; **64** (1968), 489]。更新的,由 André Joyal 所建立的数学物种理论 (theory of species) [*Advances in Math.* **42** (1981), 1 ~ 82] 导出了一种高水平的观点,在这种观点之下,生成函数的代数操作直接对应于结构的组合性质:由 F. Bergeron, G. Labelle 以及 P. Leroux 所著的 *Combinatorial Species and Tree-Like Structures* (Cambridge Univ. Press, 1998),介绍了这优美和有启发的理论的许多例子,推广了上面所

推导的许多公式。

习 题

1. [*M20*](G. Pólya) 证明

$$A(z) = z \cdot \exp(A(z) + \frac{1}{2}A(z^2) + \frac{1}{3}A(z^3) + \cdots)$$

[提示:取(3)的对数。]

2. [*HM24*](R. Otter) 证明数 a_n 满足下列条件:

$$na_{n+1} = a_1 s_{n1} + 2a_2 s_{n2} + \cdots + na_n s_{nn}$$

其中

$$s_{nk} = \sum_{1 \leq j \leq n/k} a_{n+1-jk}$$

(这些公式对于 a_n 的计算是很有用的,因为 $s_{nk} = s_{(n-k)k} + a_{n+1-k}$。)

3. [*M40*] 编写对于 $n \leq 100$ 的确定具有 n 个顶点的(不带标号的)自由树和有向树个数的程序。(使用习题 2 的结果。)考察这些数的算术性质;关于它们的素因子,或者它们对于 p 取模的余,有什么话可说?

▶ 4. [*HM39*](G. Pólya, 1937) 利用复变函数理论,确定有向树数目的渐近值如下:

a) 证明在 0 和 1 之间有一个实数 α,对于它 $A(z)$有收敛于 α 的半径,而且对于所有使得 $|z| \leq \alpha$ 的复数 z, $A(z)$绝对收敛,并有极大值 $A(\alpha) = a < \infty$。[提示:当幂级数有非负系数时,它或者是整函数或者有一个正的实奇异点;并使用习题 1 中的恒等式证明当$z \to \alpha-$时,$A(z)/z$ 有界。]

b)设

$$F(z,w) = \exp(zw + \frac{1}{2}A(z^2) + \frac{1}{3}A(z^3) + \cdots) - w$$

证明在$(z,w) = (\alpha, a/\alpha)$的一个邻域内,$F(z,w)$在每个分开的变量下解析。

c)证明在点$(z,w) = (\alpha, a/\alpha)$内,$\partial F/\partial w = 0$;因此 $a = 1$。

d)在点$(z,w) = (\alpha, 1/\alpha)$处,证明

$$\frac{\partial F}{\partial z} = \beta = \alpha^{-2} + \sum_{k \geq 2} \alpha^{k-2} A'(\alpha^k), \quad 且 \frac{\partial^2 F}{\partial w^2} = \alpha$$

e)当$|z| = \alpha$ 和$z \neq \alpha$ 时,证明$\partial F/\partial w \neq 0$;因此 $A(z)$仅在$|z| = \alpha$ 处有一个奇异点。

f)证明有一个大于$|z| < \alpha$ 的区域,其中

$$\frac{1}{z}A(z) = \frac{1}{\alpha} - \sqrt{2\beta(1 - z/\alpha)} + (1 - z/\alpha)R(z)$$

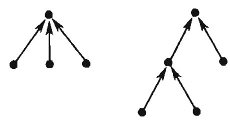
其中 $R(z)$是$\sqrt{z-\alpha}$的解析函数。

g)证明结果

$$a_n = \frac{1}{\alpha^{n-1} n}\sqrt{\beta/2\pi n} + O(n^{-5/2}\alpha^{-n})$$

[注:$1/\alpha \approx 2.955765285652$ 及 $\alpha\sqrt{\beta/2\pi} \approx 0.439924012571$。]

▶ 5. [*M25*](A.Cayley) 设 c_n 是有 n 个叶(即入度为零的顶点)以及隔一个顶点至少有两个子树的(不带标号的)有向树的个数。因此,由于两棵树



的特性,$c_3 = 2$。试求对于生成函数

$$C(z) = \sum_n c_n z^n$$

的类似于(3)的公式。

6. [*M25*] 设"有向二叉树"是这样的有向树,其中每个顶点有 2 或更少的入度。试求一个相当简单的关系,它定义带有 n 个顶点的不同有向二叉树个数的生成函数 $G(z)$,并求其开头的一些值。

7. [*HM40*] 试得出习题 6 的数的渐近值(见习题 4)。

8. [*20*] 按照等式(9),有六个带有六个顶点的自由树。试把它们画出来,并指出它们的形心。

9. [*M20*] 在一个自由树中,一个顶点至多有一个子树可包含形心,从这一事实出发,证明在自由树中至多有两个形心,而且如果有两个,则它们必相邻。

▶ 10.[*M22*] 证明具有 n 个顶点和两个形心的自由树是由各有 $n/2$ 个顶点的两个自由树组成的,两者通过一个边连接。反之,如果有 m 个顶点的两个自由树由一边连接,我们就得到由 $2m$ 个顶点和两个形心组成的自由树。

▶ 11.[*M28*] 正文导出有 n 个节点的不同二叉树个数(14)。推广它,求有 n 个节点的不同 t 叉树的个数。(见习题 2.3.1-35;t 叉树或者为空,或者由一个根和 t 个不相交的 t 叉树组成。)提示:使用 1.2.9 小节的等式(21)。

12.[*M20*] 通过使用行列式和习题 2.3.4.2-19 的结果(也见习题 1.2.3-36),试求有 n 个顶点的带标号有向树的个数。

13.[*15*] 在顶点{1,2,…,10}上的哪些有向树具有典型表示 3,1,4,1,5,9,2,6,5?

14.[*10*] 真或假:在有向树的典型表示中的最后的项 $f(V_{n-1})$,总是该树的根。

15.[*21*] 试讨论存在于(如果有的话)2.2.3 小节的拓扑排序算法和有向树的典型表示之间的关系。

16.[*25*] 试设计一个(尽可能有效的)算法,它把有向树的典型表示,转换成使用 PARENT 链接的传统的计算机表示。

▶ 17.[*M26*] 令 $f(x)$是整数值函数,其中对于所有整数 $1 \leqslant x \leqslant m$,有 $1 \leqslant f(x) \leqslant m$。如果对于某个 $r, s \geqslant 0$,$f^{[r]}(x) = f^{[s]}(y)$,其中 $f^{[0]}(x) = x$ 和 $f^{[r+1]}(x) = f(f^{[r]}(x))$,则定义 $x \equiv y$。通过使用本小节中的枚举方法,证明对于所有的 x 和 y,使得 $x \equiv y$ 的函数的个数是 $m^{m-1}Q(m)$,其中 $Q(m)$是在 1.2.11.3 小节中定义的函数。

18.[*24*] 试证明下列方法是定义从 1 到 n 的$(n-1)$元组与有 n 个带标号顶点的有向树之间的一一对应的另一种方法:设在递升顺序下树的叶是 $V_1, \cdots, V_k$,设$(V_1, V_{k+1}, V_{k+2}, \cdots, V_q)$是从 V_1 到根的通路,并写下顶点 $V_q, \cdots, V_{k+2}, V_{k+1}$。然后设$(V_2, V_{q+1}, V_{q+2}, \cdots, V_r)$是从 V_2 出发的

最短有向通路，使得 V_r 已被写下，并且写下 $V_r,\cdots,V_{q+2},V_{q+1}$。然后设$(V_3,V_{r+1},\cdots,V_s)$是从 V_3 出发的最短有向通路使得 V_s 已被写下，并写下 $V_s,\cdots,V_{r+1}$，等等。例如，树(17)编码为 3,1,3,3,5,10,5,10,1。试证明这个过程是可逆的，特别地，画出具有顶点{1,2,…,10}和表示 3,1,4,1,5,9,2,6,5 的有向树。

19.[*M24*] 有多少个有 n 个顶点的，且其中 k 个为叶(即入度为零)的不同的带标号的有向树？

20.[*M24*](J. Riordan) 有多少个有 n 个顶点，且其中 k_0 个有入度 0，k_1 个有入度 1，k_2 个有入度 2，…的不同的带标号的有向树？(注意必须有 $k_0+k_1+k_2+\cdots=n$ 及 $k_1+2k_2+3k_3+\cdots=n-1$。)

▶ 21.[*M21*] 试枚举带标号的有向树的个数，在这些树中，每个顶点有 0 或 2 的入度。(见习题 20 和习题 2.3-20。)

22.[*M20*] 有多少种可能的有 n 个顶点的带标号自由树？(换言之，如果给了 n 个顶点，依赖于$\binom{n}{2}$个可能的边中哪些被加入到图中，便有 $2^{\binom{n}{2}}$ 个可能的有这些顶点的图；这些图中有多少个是自由树？)

23.[*M21*] 对于 n 个带标号的顶点来说，有多少种可能的有序树？(给出涉及阶乘的简单公式。)

24.[*M16*] 有顶点 1,2,3,4 和以 1 为根的所有带标号的有向树已在(15)中示出。如果我们列出具有此顶点和此根的带标号的有序树，将会有多少种这样的树？

25.[*M20*] 等式(18)和(19)出现的量 $r(n,q)$的值是多少？(请给出显式的公式；正文中只提及 $r(n,n-1)=n^{n-2}$。)

26.[*20*] 借助于在本小节结尾处的记号，画出类似于(23)的((3,2,4),(1,4,2))构造，并求出对应于有 $t=8$ 的典型表示，即颜色序列“红，黄，蓝，红，黄，蓝，红，蓝，蓝”和“红，黄，蓝，黄，黄，蓝，黄”以及下标序列 3;1,2,2,1;2,4 的数 k。

▶ 27.[*M28*] 设 $U_1,U_2,\cdots,U_p,\cdots,U_q;V_1,V_2,\cdots,V_r$ 是一个有向图的顶点，其中 $1\leqslant p\leqslant q$。设 f 是从集合$\{p+1,\cdots,q\}$到集合$\{1,2,\cdots,r\}$的任何函数，并设这个有向图恰含 $q-p$ 条边，对于 $p<k\leqslant q$ 它们是从 U_k 到 $V_{f(k)}$。证明加上另外的 r 条有向边，即从每个 V 到相应 U 的有向边，使得所得到的有向图不含有向回路，这样的方式数，是 $q^{r-1}p$。通过推广典型表示方法来证明这一点；即，建立加上 r 个进一步的边的所有这样的方式和所有整数序列 $a_1,a_2,\cdots,a_n$ 的集合之间的一一对应，其中对于 $1\leqslant k<r$，$1\leqslant a_k\leqslant q$ 和 $1\leqslant a_r\leqslant p$。

28.[*M22*](双部树) 使用习题 27 的结果来枚举在顶点 $U_1,\cdots,U_m,V_1,\cdots,V_n$ 上的带标号自由树的个数，使得对于某个 j 和 k，每个边将 U_j 连到 V_k。

29.[*HM26*] 证明如果 $E_k(r,t)=r(r+kt)^{k-1}/k!$，且如果 $zx^t=\ln x$，则对于固定的 t 和对于充分小的$|z|$和$|x-1|$有

$$x^r=\sum_{k\geqslant 0}E_k(r,t)z^k$$

[使用在等式(19)下面的讨论中 $G_m(z)=G_1(z)^m$ 的事实。]在这个公式中，r 代表任意实数。[注：作为这个公式的结果，我们有恒等式

$$\sum_{k=0}^{n}E_k(r,t)E_{n-k}(s,t)=E_n(r+s,t)$$

这导出 1.2.6 小节的等式(16),即阿贝尔二项式定理。并同该小节的等式(30)作比较。]

30.[*M23*] 设 $n, x, y, z_1, \cdots, z_n$ 是正整数。考虑 $x + y + z_1 + \cdots + z_n + n$ 个顶点 r_i, s_{jk}, t_j($1 \leq i \leq x + y, 1 \leq j \leq n, 1 \leq k \leq z_j$)的一个集合,其中对于所有 j 和 k,从 s_{jk} 到 t_j,已画出有向边。按照习题 27,共有 $(x + y)(x + y + z_1 + \cdots + z_n)^{n-1}$ 种方法来描出从 $t_1, \cdots, t_n$ 的每一个到其它顶点的有向边,使得得到的有向图不含有向回路。利用这一点来证明 Hurvitz 对于二项式定理的推广:

$$\sum x(x + \epsilon_1 z_1 + \cdots + \epsilon_n z_n)^{\epsilon_1 + \cdots + \epsilon_n - 1} y(y + (1 - \epsilon_1) z_1 + \cdots + (1 - \epsilon_n) z_n)^{n-1-\epsilon_1-\cdots-\epsilon_n} =$$

$$(x + y)(x + y + z_1 + \cdots + z_n)^{n-1}$$

其中求和是对等于 0 或 1 的 $\epsilon_1, \cdots, \epsilon_n$ 的所有 2^n 种选择进行的。

31.[*M24*] 对于有序树求解习题 5;即,导出有 n 个终节点和没有度数为 1 的节点的不带标号的有序树个数的生成函数。

32.[*M37*] (A. Erdélyi 和 I. M. H. Etherington, *Edinburgh Math. Notes* **32** (1941), 7 ~ 12) 对于有 n_0 个 0 度节点,n_1 个 1 度节点,$\cdots$,n_m 个 m 度节点,且无比 m 更高的度数的节点,有多少个(有序和不带标号的)树;(借助于阶乘,可以给出这个问题的显式解,因而相当大地推广了习题 11 的结果。)

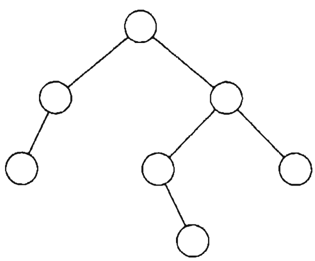
▶ 33.[*M28*] 正文给出基于某些有向森林的枚举公式,方程 $w = y_1 e^{z_1 w} + \cdots + y_r e^{z_r w}$ 的一个显式幂级数解。类似地,证明习题 32 的枚举公式导出方程

$$w = z_1 w^{e_1} + z_2 w^{e_2} + \cdots + z_r w^{e_r}$$

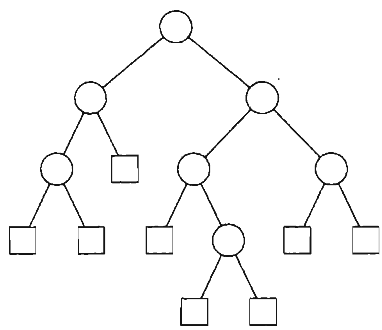
的一个显式幂级数解,并把 w 表达为 $z_1, \cdots, z_r$ 的幂级数。(这里 $e_1, \cdots, e_r$ 是固定的非负整数,其中至少有一个为零。)

2.3.4.5　通路长度　在算法分析中,一个树的"通路长度"的概念是极为重要的,因为这个量经常直接同执行时间有关。我们主要关心二叉树,因为它们同实际的计算机表示关系非常密切。

在以下的讨论中,在原来的树中出现空子树的每个地方,我们都加上特殊节点,由此推广每个二叉树的框图,使得



变成



(1)

后者称为*扩充的二叉树*。在以这种方法加上方形节点之后,这个结构处理起来更方便,因此在稍后几章中我们将经常碰到扩充的二叉树。显然,每一个圆圈节点有两个儿子,而每个方形节点没有。(试同习题 2.3-20 作比较。)如果有 n 个圆圈节点和 s 个方形节

点，我们就有 $n+s-1$ 个边(因为框图是一个自由树)；以另一种方法来计数，即通过儿*子个数来计算，我们看到，有* $2n$ 条边。因此显然

$$s = n + 1 \tag{2}$$

换句话说，刚刚加上去的“外部”节点的个数比我们原来有的“内部”节点数多 1。(关于另一种证明方法，见习题 2.3.1-14。)即使当 $n=0$ 时，公式(2)仍然正确。

假定已经以这种方式对二叉树作了扩充。**树的外部通路长度** E 被定义为从根到每个节点——取遍所有外部(方形)节点——的通路长度之和。**内部通路长度** I 是对内部(圆圈)节点求和的同样的量。在(1)中，外部通路长度是

$$\begin{aligned} E &= 3+3+2+3+4+4+3+3 \\ &= 25 \end{aligned}$$

而内部通路长度是

$$\begin{aligned} I &= 2+1+0+2+3+1+2 \\ &= 11 \end{aligned}$$

这两个量总是通过公式

$$E = I + 2n \tag{3}$$

相关联，其中 n 是内部节点的个数。

为了证明公式(3)，考虑从根处删去在距离为 k 处的内部节点 V，其中 V 的两个儿子都是外部节点。数量 E 减少 $2(k+1)$，因为 V 的儿子已被删去，然后它增加 k，因为 V 变成外部节点；因此 E 中的净变化是 $-k-2$，I 的净变化是 $-k$，由归纳法可得出(3)。

不难看出，当我们有带有线性结构的一个退化树时，内部通路长度(且因此外部通路长度亦然)是最大的；在此情况下，内部通路长度是

$$(n-1)+(n-2)+\cdots+1+0 = \frac{n^2-n}{2}$$

可以证明，对于所有的二叉树，“平均的”通路长度实质上同 $n\sqrt{n}$ 成比例(见习题 5)。

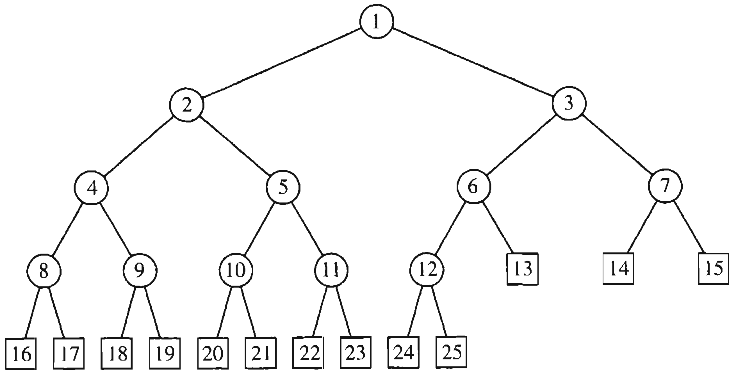
现在考虑构造有极小通路长度的有 n 个节点的二叉树的问题。这样一种树是重要的，因为它将极小化各种算法的计算时间。显然，从根处出发只有一个节点(即根)可以有距离 0；至多两个节点可以有距离 1，至多有四个节点可以有距离 2，等等。因此，*内部通路长度总是至少和下列数组的前* n *项之和一样大*：

$$0,1,1,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,4,\cdots$$

这个和是 $\sum_{k=1}^{n}\lfloor \lg k\rfloor$，由习题 1.2.4-42 我们知道它是

$$(n+1)q - 2^{q+1} + 2, \quad q = \lfloor \lg(n+1)\rfloor \tag{4}$$

最优值(4)是 $n\lg n + O(n)$，因为 $q = \lg n + O(1)$；显然这在像(5)这样的树中被达到(对于 $n=12$ 示出)。

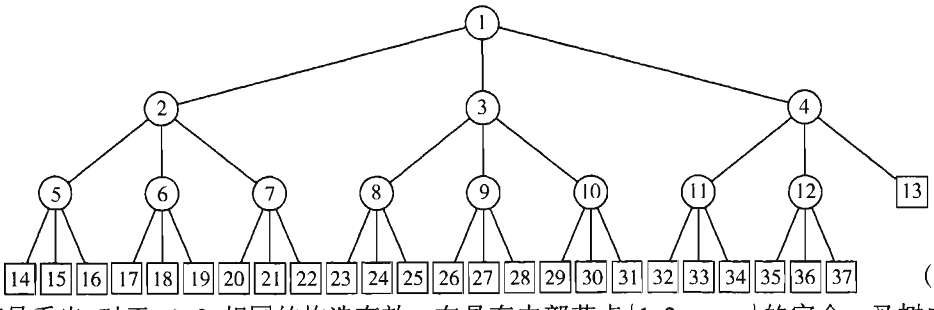


(5)

像(5)这样的树称为有 n 个内部节点的完全二叉树。一般情况下,我们可对内部节点编号 $1,2,\cdots,n$;这个编号有很有用的性质,即节点 k 的父亲是$\lfloor k/2\rfloor$,而节点 k 的儿子是节点 $2k$ 和 $2k+1$。外部节点的编号是从 $n+1$ 直到 $2n+1$,包括这两者在内。

由此得出,完全二叉树可以简单地以顺序的内存位置表示,且其结构隐含在节点的位置中(而不在链接中)。完全二叉树或者明显地或者隐含地出现在许多重要的计算机算法当中,所以读者应对它予以特别的注意。

这些概念对于三叉,四叉以及更高阶的树有重要的推广。我们把 t 叉树定义为或者为空,或者由一个根和 t 个有序的不相交的 t 叉树组成。(这推广了在 2.3 节中的二叉树的定义。)具有 12 个内部节点的完全三叉树是



(6)

容易看出,对于 $t\geq 2$,相同的构造有效。在具有内部节点$\{1,2,\cdots,n\}$的完全 t 叉树中,节点 k 的父亲节点是

$$\lfloor (k+t-2)/t\rfloor = \lceil (k-1)/t\rceil$$

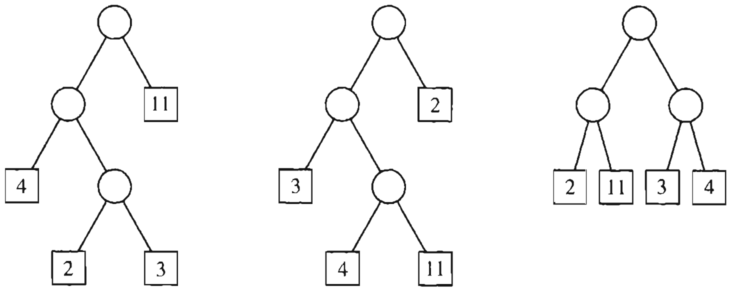
而节点 k 的儿子节点是

$$t(k-1)+2,\quad t(k-1)+3,\quad \cdots,\quad tk+1$$

这棵树在具有 n 个内部节点的所有 t 叉树中有极小内部通路长度;习题 8 证明它的内部通路长度是

$$\left(n+\frac{1}{t-1}\right)q-\frac{(t^{q+1}-t)}{(t-1)^2},\quad q=\lfloor \log_t((t-1)n+1)\rfloor \tag{7}$$

如果我们稍微转移观点,这些结果有另一种重要的推广。假设给了我们 m 个实数 $w_1,w_2,\cdots,w_m$;问题是求具有 m 个外部节点的扩充的二叉树,并且以这样一种方式把数 $w_1,\cdots,w_m$ 同这些节点关联起来,即要使和数 $\sum w_jl_j$ 是极小的,其中 l_j 是从根出发的通路长度,而且和数是对所有外部节点来取的。例如,如果给定的数是 2,3,4,11,我们可以形成像以下三种这样的扩充的二叉树



(8)

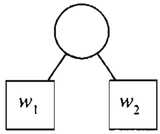
这里"带权的"通路长度 $\sum w_jl_j$ 分别是 34,53 和 40。(因此当权是 2,3,4 和 11 时,完全平衡树并不给出极小权通路长度,尽管我们已经看到,在 $w_1=w_2=\cdots=w_m=1$ 的特殊情况下它确实给出极小值。)

在与不同的计算机算法相关联时,出现了带权通路长度的若干个解释;例如,我们可以应用它到长度分别为 $w_1,w_2,\cdots,w_m$ 的排序序列的合并(参见第 5 章)。这个思想最直截了当的应用之一是把一个二叉树作为一般的查找过程,其中我们在根处开始,而后作某个检测;检测的结果把我们送到两个分支之一,在那里我们可以作进一步的检测,等等。例如,如果我们要判断四个不同的选择中哪一个是对的,而且这些可能性以分别的概率 $\frac{2}{20},\frac{3}{20},\frac{4}{20}$ 和 $\frac{11}{20}$ 为真,则极小化带权通路长度的树将组成最优查找过程。[这些是(8)中所示的权,乘上一个纯量因子。]

用于求出具有极小带权通路长度的树的漂亮的算法是由 D. Huffman(哈夫曼)发现的[*Proc. IRE* **40** (1951),1098～1101]:首先求最小值的两个 w,比如说 w_1 和 w_2,然后对于 $m-1$ 个权 $w_1+w_2,w_3,\cdots,w_m$,求解问题,并且把这个解中的节点

$$\boxed{w_1+w_2} \tag{9}$$

用



(10)

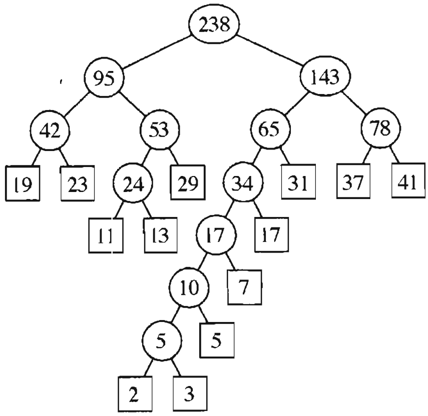
代替。

作为哈夫曼方法的一个例子,让我们来求权 2,3,5,7,11,13,17,19,23,29,31,37,41 的最优树。首先我们组合 2+3,并寻找 5,5,7,⋯,41 的解,然后组合 5+5,等等。这个

计算可概述如下：

$\underline{2}$	$\underline{3}$	5	7	11	13	17	19	23	29	31	37	41
	$\underline{5}$	$\underline{5}$	7	11	13	17	19	23	29	31	37	41
		$\underline{10}$	$\underline{7}$	11	13	17	19	23	29	31	37	41
			17	$\underline{11}$	$\underline{13}$	17	19	23	29	31	37	41
			$\underline{17}$		24	$\underline{17}$	19	23	29	31	37	41
					24	34	$\underline{19}$	$\underline{23}$	29	31	37	41
					$\underline{24}$	34		42	$\underline{29}$	31	37	41
						$\underline{34}$		42	53	$\underline{31}$	37	41
								42	53	65	$\underline{37}$	$\underline{41}$
								$\underline{42}$	$\underline{53}$	65		78
									95	$\underline{65}$		$\underline{78}$
									$\underline{95}$			$\underline{143}$
												238

因此对应于哈夫曼构造的树如下：



(11)

(在圆圈节点内的数示出这个树和我们的计算之间的对应，也见习题 9。)

通过对 m 的归纳法不难证明，这个方法事实上确实极小化带权通路长度。假设我们有 $w_1 \leqslant w_2 \leqslant w_3 \leqslant \cdots \leqslant w_m$，其中 $m \geqslant 2$，而且假设给了我们极小化带权通路长度的树。(这种树肯定存在，因为只可能有有限多个有 m 个终端节点的二叉树。)令 V 是从根出发有极大距离的内部节点。如果 w_1 和 w_2 不是已附加给 V 的儿子的权，我们可以把它们同已经在那里的值交换；这样的交换不增加带权通路的长度。因此存在极小化带权通路长度的树并且包含子树(10)。现在容易证明，这种树的带权通路长度是极小的，当且仅当以(9)代替(10)的树对于权 $w_1 + w_2, w_3, \cdots, w_m$ 来说有极小通路长度。(见习题 9。)

每次这个构造把两个权组合在一起，如果给定的权是非负的，则它们至少和以前组合的权一样大。这意味着，假定已经把给定的权排序成非递减的顺序，则有简洁的方式来寻找出哈夫曼树：我们只需维护两个队列，一个含原来的权，另一个包含已组合了的

权。在每一步骤中,最小的还未用过的权将出现在队列之一的前端,所以我们不用加以搜索。参见习题 13,该习题表明,即使权可能为负时,同样的思想也仍有效。

一般说来,有许多极小化 $\sum w_j l_j$ 的树。如果在上一段落中概述的算法在相同的情况下,总是使用原来的权而不是组合了的权,则它构造的树在极小化 $\sum w_j l_j$ 的所有树当中有最小 $\max l_j$ 和最小的 $\sum l_j$ 的值。如果权都为正,这个树实际上对所有这样的树,对任何凸函数 f 都极小化 $\sum w_j f(l_j)$。[参见 E.S.Schwartz, *Information and Control* **7** (1964), 37~44; G.Markowsky, *Acta Informatica* **16** (1981), 363~370。]

哈夫曼方法既可推广到二叉树,也可推广到 t 叉树(见习题 10)。6.2.2 小节讨论哈夫曼方法的另一重要推广。5.3.1,5.4.9 和 6.3 等节还有关于通路长度进一步的讨论。

习　题

1.[*12*] 除了完全树(5)之外,还有其它任何具有 12 个内部节点和极小通路长度的二叉树吗?

2.[*17*] 画出具有终节点且包含权 1,4,9,16,25,36,49,64,81,100 和极小通路长度的扩展二叉树。

▶ 3.[*M24*] 有 m 个外部节点的扩充二叉树确定长度为 $l_1, l_2, \cdots, l_m$ 的一组通路,它们描述从根开始到各个外部节点的通路的长度。反之,如果给定一组数 $l_1, l_2, \cdots, l_m$,是否总有可能来构造这样一个扩充二叉树,其中这些数是在某个顺序下的通路长度?试证明这是可能的,当且仅当 $\sum_{j=1}^{m} 2^{-l_j} = 1$。

▶ 4.[*M25*] (E.S.Schwartz 和 B.Kallick) 假定 $w_1 \leqslant w_2 \leqslant \cdots \leqslant w_m$。证明存在可极小化 $\sum w_j l_j$ 的一个扩充二叉树。而且对于它在自左至右的顺序下终端节点分别包含值 $w_1, w_2, \cdots, w_m$。[例如,树(11)不满足这个条件,因为诸权是以 19,23,11,13,29,2,3,5,7,17,31,37,41 的顺序出现的。我们寻求其权以递增顺序出现的树,而且对于哈夫曼构造来说这不总发生。]

5.[*HM26*] 令

$$B(w,z) = \sum_{n,p \geq 0} b_{np} w^p z^n$$

其中 b_{np} 是具有 n 个节点和内部通路长度 p 的二叉树的数目。[于是,

$$B(w,z) = 1 + z + 2wz^2 + (w^2 + 4w^3)z^3 + (4w^4 + 2w^5 + 8w^6)z^4 + \cdots$$

$B(1,z)$ 是在 2.3.4.4 小节中等式(13)的函数 $B(z)$。]

a)推广 2.3.4.4-(12),求表征 $B(w,z)$ 的函数关系。

b)假定 $\frac{1}{n+1}\binom{2n}{n}$ 个树中的每一个在概率上都是相等的,请使用 a)的结果来确定具有 n 个节点的二叉树的平均内部通路长度。

c)求这个量的渐近值。

6.[*16*] 如果像在(1)中那样以方形节点对 t 叉树进行扩充,对应于等式(2)的方形节点和圆圈节点之间的关系是什么?

7.[*M21*] t 叉树中的外部通路长度与内部通路长度之间有什么关系?(见习题 6;要求对方程(3)加以推广。)

8.[*M23*] 证明等式(7)。

9.[*M21*] 出现在(11)中的圆圈节点的数等于对应的子树的外部节点中的权之和。试证明在圆圈节点中所有值之和等于带权通路长度。

▶ 10.[*M26*] (D.Huffman) 给定非负的权 $w_1,w_2,\cdots,w_m$,说明如何构造带有极小带权通路长度的 t 叉树。对于权 1,4,9,16,25,36,49,64,81,100 构造一棵最优三叉树。

11.[*16*] 在完全二叉树(5)和在习题 2.3.1-5 中描述的二叉树的"杜威十进记号"之间是否有任何关系?

▶ 12.[*M20*] 假设在二叉树中,以每个节点都是同等可能的条件随机选择一个节点。说明以该节点为根的子树的平均大小同树的通路长度有关。

13.[*22*] 试设计一种算法,以 m 个权 $w_1\leqslant w_2\leqslant\cdots\leqslant w_m$ 开始,构造有极小带权通路长度的扩充二叉树。以三个数组来表示最后的树:

$$A[1]\cdots A[2m-1],\quad L[1]\cdots L[m-1],\quad R[1]\cdots R[m-1]$$

这里 $L[i]$和 $R[i]$指向内节点 i 的左和右的儿子,根是节点 1,而 $A[i]$是节点 i 的权。原来的权应作为外部节点的权 $A[m],\cdots,A[2m-1]$出现。你的算法应当做少于 $2m$ 次的权的比较。警告:给定的权有些乃至全部都可能为负的!

14.[*25*] (胡德强和 A.C.Tucker) 哈夫曼算法执行了 k 步之后,迄今被组合了的节点形成 $m-k$个扩充二叉树的森林。证明,在有给定的权的所有 $m-k$ 个扩充二叉树的所有森林当中,这个森林有最小的总的带权通路长度。

15.[*M25*] 证明类似哈夫曼的算法可求出符合以下条件的扩充二叉树:(a)极小化 $\max(w_1+l_1,\cdots,w_m+l_m)$;(b)给定 $x>1$,极小化 $w_1x^{l_1}+\cdots+w_mx^{l_m}$。

16.[*M25*] (黄光明) 令 $w_1\leqslant\cdots\leqslant w_m$ 和 $w'_1\leqslant\cdots\leqslant w'_m$ 是两组权且

$$\sum_{j=1}^{k}w_j\leqslant\sum_{j=1}^{k}w'_j,\quad 对于 1\leqslant k\leqslant m$$

试证明极小带权通路长度满足 $\sum_{j=1}^{m}w_jl_j\leqslant\sum_{j=1}^{m}w'_jl'_j$。

17.[*HM30*] (C.R.Glassey 和 R.M.Karp) 令 $s_1,\cdots,s_{m-1}$是在构造的顺序下,由哈夫曼算法所形成的扩充二叉树的内部(圆圈)节点内的数。令 $s'_1,\cdots,s'_{m-1}$是在同一权的集合$\{w_1,\cdots,w_m\}$上的任何扩充二叉树的内部节点的权,它们是以每个非根的内部节点出现在它的父亲节点之前的任何顺序列出的。(a)证明对于 $1\leqslant k<m$,$\sum_{j=1}^{k}s_j\leqslant\sum_{j=1}^{k}s'_j$。(b)对于每个非递减的凹函数 f,即满足 $f'(x)\geqslant0$ 和 $f''(x)\leqslant0$ 的每个函数 f,(a)的结果等于

$$\sum_{j=1}^{m-1}f(s_j)\leqslant\sum_{j=1}^{m-1}f(s'_j)$$

[参见 Hardy,Littlewood 和 Pólya,*Messenger of Math.* **58** (1929),145 ~ 152。]使用这一事实证明,给定满足 $\Delta f(n)=f(n+1)-f(n)\geqslant0$ 和 $\Delta^2f(n)=\Delta f(n+1)-\Delta f(n)\leqslant0$ 的性质的任何函数 $f(n)$,在递推式 $F(n)=f(n)+\min\limits_{1\leqslant k<n}(F(k)+F(n-k))$,$F(1)=0$ 中,极小值总是当 $k=2^{\lceil\lg(n/3)\rceil}$时出现。

***2.3.4.6 历史和文献** 树当然在创世的第三天就已经存在了,而且历经岁月树结构(特别是家族树)已经被普遍使用。树概念作为一种形式化定义的数学实体似乎首先出现在基尔霍夫的工作中[*Annalen der Physik und Chemie* **72** (1847),497 ~ 508 ,英译见 *IRE Transactions* **CT-5** (1958), 4 ~ 7]。基尔霍夫使用自由树,在一个电气网络中,联系以他的名字命名的法则求一组基本回路,实际上就像我们在 2.3.4.1 小节所做的那样。这

个概念大约在同一时期出现在 K.G.Chr.von Staudt 的著作*Geometrie der Lage*.（20～21）中。“树”这个名字以及涉及大多数树的枚举的许多结果十年之后出现于 Arthur Cayley 的一系列论文中[请见*Collected Mathematical Papers* **3**（1857），242～246；**4**（1859），112～115；**9**（1874），202～204；**9**（1875），427～460；**10**（1877），598～600；**11**（1881），365～367；**13**（1889），26～28]。Cayley 不知道基尔霍夫和 von Staudt 以前的工作；他的研究以代数公式的结构开始，而且过后这些工作主要是受到化学中异构体问题应用的鼓舞与激励。树结构也被 C.W.Borchardt [*Crelle* **57**（1860），111～121]，J.B.Listing [*Göttinger Abhandlungen*，Math.Classe，**10**（1862），137～139]以及 C.Jordan[*Crelle* **70**（1869），185～190]等独立地研究。

“无穷性引理”首先是由 Dénes König 系统阐述的[*Fundamenta Math*. **8**（1926），114～134]，他在其经典著作*Theorie der endlichen und unendlichen Graphen*（Leipzig：1936），第 6 章中把它放在显赫的位置上。类似的叫做“趣味定理”的结果稍早出现在 L.E.J.Brouwer 的著作[*Verhandelingen Akad.Amsterdam* **12**（1919），7]中，但这涉及强得多的假设；请见 A.Heyting，*Intuitionism*（1956），3.4 节关于 Brouwer 著作的讨论。

2.3.4.4 小节中关于枚举不带标号的有向树的公式(3)，是由 Cayley 在他关于树的第一篇论文中给出的。在他的第二篇论文中他枚举了不带标号的有向树；在几何中的一个等价问题(见习题 1)已经由 J.von.Segner 和 L.Euler 先于此 100 年提出并解决了[*Novi Commentarii Academiæ Scientiarum Petropolitanæ* **7**（1758～1759），13～15，203～210]，而且它是 G.Lamé，E.Catalan，O.Rodrigues 及 J.Binet 在*Journal de Mathématiques* **3**，**4**（1838，1839）中的七篇论文的主题；另外的参考文献出现于习题 2.2.1-4 的答案中。相应的数现在普遍称为“Catalan(卡塔兰)数”。中国的蒙族数学家明安图(Ah-T'u Ming)在 1750 年之前在他关于无穷级数的研究中就碰到了卡塔兰数，但他没有把卡塔兰数同树或其它有关课题联系起来[参见罗见今，《内蒙古大学自然科学学报》**19**（1978），239～245；*Combinatorics and Graph Theory*（World Scientific Publishing，1993），68～70]。卡塔兰数出现在许多不同的范畴中；Richard Stanley 在他的大作*Enumerative Combinatorics* **2**（Cambridge Univ.Press，1999)第 6 章中，说明了 60 种以上的应用范畴。也许最令人惊奇的是卡塔兰数同某些数的排列的关联，H.S.M.Coxetor 称这些排列为“中楣模式”，这是由于它们的对称性所致；请见习题 4。

关于带标号自由树数目的的公式 n^{n-2}(习题 2.3.4.2-28)，作为对于某个行列式的计算的副产品，是由 J.J.Sylvester 发现的[*Quart. J. Pure and Applied Math*. **1**（1857），55～56]。Cayley 于 1889 年对这个公式给出独立的推导[请见上面的参考文献]；他的讨论非常模糊，提示了带标号有向树和数的($n-1$)元组之间的关系。Heinz Prüfer 最先发表了揭示这样一种关系的明确的对应[*Arch.Math. und Phys*. **27**（1918），142～144]，而且完全独立于 Cayley 以前的工作。关于这个主题已发表了大量文献，J.W.Moon 在*Counting Labelled Trees* 中优美地综述了经典结果(Montreal：Canadian Math.Congress，1970)。

G Pólya 在*Acta Math*. **68**（1937），145～253 中，发表了关于树以及其它种类的组合结构的枚举的非常重要的论文。关于图的枚举问题的讨论以及早期论著中的杰出文献目录，请见 Frank Hararg 在*Graph Theory and Theoretical Physics*（London：Academic Press，1967），

1 ~ 41 上的综述。

通过重复地组合最小的权来极小化带权通路长度的原理,是在同极小化消息长度的编码设计相联系之下,由 D. Huffman 发现的[*Proc. IRE* **40** (1952), 1098 ~ 1101]。同样的思想,也由 Seth Zimmerman 独立地发表[*AMM* **66** (1959), 690 ~ 693]。

关于树结构理论的许多值得注意的论文,与特定课题相联系,在 2.3.4.1 小节到 2.3.4.5 小节中已经都引用了。

习　题

▶ 1. [*21*] 假定多边形的边是不同的,试求具有 n 个节点的二叉树同把 $(n+2)$ 个边的凸多边形分割成为三角形的分割之间简单的一一对应关系。

▶ 2. [*M26*] 1857 年, T. P. Kirkman 猜想,在 n 边形中画 k 条不相重叠的对角线的方法数是

$$\binom{r+k}{k+1}\binom{r-3}{k}\bigg/(r+k)$$

a)推广习题 1 的对应以得到关于树枚举的等价问题。

b)通过使用习题 2.3.4.4-32 的方法证明 Kirkman 的猜想。

▶ 3. [*M30*] 考虑把凸的 n 边形的顶点分划成 k 个非空部分的所有方法,使得在一个部分中的两个顶点之间没有对角线同另一个部分的两个顶点之间的对角线相交。

a)求不相交的分划和有趣的树结构类之间的一一对应。

b)给定 n 和 k,共有多少种这样的分划方法?

▶ 4. [*M38*] (Conway 和 Coxeter) 中楣模式是类似于下面的无穷阵列:

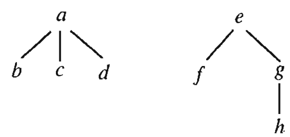
```
1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   …
  3   1   3   1   4   1   2   3   1   3   1   4   1   2   3   1   3   1   4   …
5   2   2   2   3   3   1   5   2   2   2   3   3   1   5   2   2   2   3   …
  3   3   1   5   2   2   2   3   3   1   5   2   2   2   3   3   1   5   2   …
1   4   1   2   3   1   3   1   4   1   2   3   1   3   1   4   1   2   3   …
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   …
```

其中顶部和底部的行完全由 1 组成。而每个相邻值的菱形 $a\,{}^{b}_{c}\,d$ 满足 $ad-bc=1$。试求 n 节点二叉树和 $(n+1)$ 行的正整数中楣模式之间的一一对应。

2.3.5 列表和废料收集

在 2.3 节接近开头处,我们把列表非正式地定义为"零个或多个原子或列表的有限序列"。

任何森林都是列表;例如

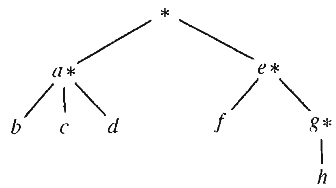


(1)

可以当做列表

$$(a:(b,c,d),e:(f,g:(h))) \tag{2}$$

而对应的列表框图将是



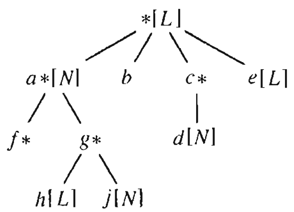
(3)

读者应回顾较早时对列表的介绍,特别是在 2.3 节开始几页中的(3),(4),(5),(6),(7)。回忆一下,在上面的(2)中,记号“$a:(b,c,d)$”意味着(b,c,d)是三个元素的列表,它已经以属性“a”加以标号。这个约定同我们总的约定是相兼容的,即一个树的每个节点可以包含除其结构联系之外的信息。然而,如同在 2.3.3 小节对于树的讨论那样,坚持所有列表都是未带标号的,使得所有信息都出现在原子上,这是十分可能的,有时也是合意的。

尽管任何森林都可看成是一个列表,但反过来就不对了。下面的列表也许比(2)和(3)要更典型,因为它表示出树结构的限制是如何被侵犯的:

$$L = (a:N,b,c:(d:N),e:L), \quad N = (f:(),g:(h:L,j:N)) \tag{4}$$

它可以画成框图如下

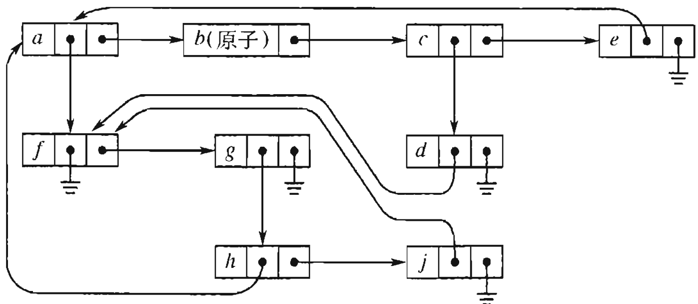


(5)

[请与 2.3-(7)中的例子加以比较。不必太在意这些框图的形式。]

如同我们可能预料的那样,在计算机内存中有许多方法来表示列表结构。这些方法通常是相同的基本主题上的变形,通过它们,我们已经使用二叉树来表示一般的树森林;一个字段,比如说 RLINK,被用来指向列表的下一个元素。而另一个字段 DLINK 可以用来指向子列表的第一个元素。通过把 2.3.2 小节所描述的内存表示进行自然扩充,我们将把列表(5)表示成(6)。

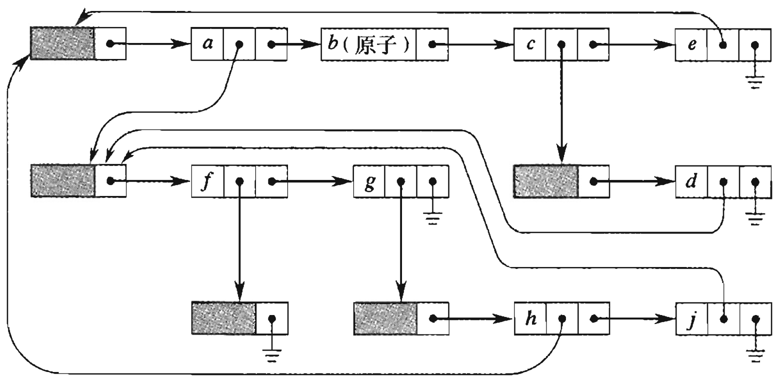
不幸的是,这个简单的想法不大适合大多数普通的列表处理应用。例如,假设有列表 $L=(A,a,(A,A))$,包含对另一个列表 $A=(b,c,d)$的三次访问。典型的列表处理操作之一是删去 A 最左边的元素,使得 A 变成为(c,d);但如果我们使用在(6)中所示技术,这就要求对 L 的表示做三次改动,因为对 A 的每个指针指向要被删去的元素 b。读者稍加思索就会相信,仅仅因为 A 的第一个元素要被删去,而要改动在每一个访问中的指针,这是极端不合要求的。(在这个例子中,我们可能试图搞点技巧,假定没有指



(6)

向 c 的指针,通过把 c 的整个元素复写到以前由 b 所占据的单元,而后删去 c 的原来元素。但当 A 失去它的最后的元素并变成空时,这个技巧就不灵了。)

由于这个原因,表示方案(6)一般为类似的另一个方案所代替,如同在 2.2.4 小节所介绍的那样,使用列表头来开始每个列表。每个列表包含称为它的列表头的一个附加节点,使得格局(6),比如说,将被表示如下:



(7)

这样的标题节点的引进在实践上并不是对内存空间的浪费,因为对于明显地还未使用的字段——框图(7)中的带阴影的区域——一般都存在许多用法。例如,这就有了空间用于一个访问计数,一个指向列表右端的指针,一个字符名,或一个"草稿"字段,等等,以帮助遍历算法。

在我们原来的框图(6)中,包含 b 的节点是一个原子,而包含 f 的节点确定空列表。两者在结构上是相同的。所以如果读者问为什么我们要费心地去谈及"原子",那将是十分正当的;不失一般性,我们本可以将列表仅定义为"零个或多个列表的有限序列",而且加上我们通常的约定,即一个列表的每个节点可以包含除了它的结构信息之外的数据。这个观点肯定是站得住脚的,并且它使"原子"的概念似乎是非常做作的。然而当考虑到计算机内存的有效使用时,有一个很好的理由要像我们所已经做的那样提及原子,因为原子不受对于列表来说需要的相同类型通用操作的支配。内存表示(6)表明,在原子节点 b 中,比起在列表节点 f 中,大概有更多的存放信息的空间;而且,当列表头节点也像在(7)中那样出现时,对于节点 b 和节点 f 来说,其存储要求之间有显著的区别。因此引进原子的概念主要是为了帮助有效地使用计算机内存。典型的列表比

起我们的例子所表示的包含多得多的原子;(4)~(7)的例子旨在说明可能的复杂性,而不是通常有的简单性。

列表实际上无异于其元素可以包含针对其它列表的指针的线性表。对于列表我们希望实施的普通操作就是人们通常希望对线性表的操作(建立,消毁,插入,删除,分裂,连接),加上主要是对于树结构感兴趣的进一步的操作(复制,遍历,输入和输出嵌入的信息)。为了这些目的在内存中表示链接线性表的三种基本技术——直接的,循环的或双重链接技术——的任何一个都可使用,而且依赖于所使用的算法而有不同程度的效率。对于这三种类型的表示,框图(7)在内存中可如下出现:

内存位置	直接链接			循环链接			双重链接				
	INFO	DLINK	RLINK	INFO	DLINK	RLINK	INFO	DLINK	LLINK	RLINK	
010:	–	头	020	–	头	020	–	头	050	020	
020:	*a*	060	030	*a*	060	030	*a*	060	010	030	
030:	*b*	原子	040	*b*	原子	040	*b*	原子	020	040	
040:	*c*	090	050	*c*	090	050	*c*	090	030	050	
050:	*e*	010	Λ	*e*	010	010	*e*	010	040	010	
060:	–	头	070	–	头	070	–	头	080	070	
070:	*f*	110	080	*f*	110	080	*f*	110	060	080	(8)
080:	*g*	120	Λ	*g*	120	060	*g*	120	070	060	
090:	–	头	100	–	头	100	–	头	100	100	
100:	*d*	060	Λ	*d*	060	090	*d*	060	090	090	
110:	–	头	Λ	–	头	110	–	头	110	110	
120:	–	头	130	–	头	130	–	头	140	130	
130:	*h*	010	140	*h*	010	140	*h*	010	120	140	
140:	*j*	060	Λ	*j*	060	120	*j*	060	130	120	

这里“LLINK”被用做指向双重链接表示中的左边的指针。在所有三种形式下 INFO 和 DLINK 字段是相同的。

在这里没有必要重复在任何这三种形式之下对列表进行操作的算法,因为我们已经讨论这些思想好多次了。然而以下关于列表的要点必须加以说明,因为这不同于以前处理过的较简单的特殊情况:

1)在上面的内存表示中隐含了,原子节点不同于非原子节点;而且,当使用循环链接列表或双重链接列表时,希望把表头节点同其它类型的节点区别开来,作为遍历列表时的辅助。因此,每个节点一般地都包含一个 TYPE 字段,指出这个节点表示什么类型的信息。这个 TYPE 字段通常用来区分各种类型的原子(例如:字符型,整型或浮点量,以供对数据进行操作或显示数据时使用)。

2)对于在 MIX 计算机上的一般列表操作,节点格式可以以下列两种方法之一来设计。

a)假定所有 INFO 都以原子出现,可能的单字格式为

S	T	REF	RLINK

(9)

S(符号):在废料收集中使用的标记位(参见下述)。

T(类型):对于列表头,T=0;对于子列表元素,T=1;对于原子,T>1。

REF:当 T=0 时,REF 是访问计数(参见下述);当 T=1 时,REF 指向问题中的子列表的列表头;当 T>1 时,REF 指向包含有标记位和五个字节的原子信息的节点。

RLINK:如同(8)中那样指向直接的或循环链接的指针。

b)可能的两字格式为

S	T	LLINK	RLINK
INFO			

(10)

S,T:与(9)中相同。

LLINK,RLINK:与(8)中一样,用于双重链接的通常的指针。

INFO:同这个节点相关联的全字信息;对于表头节点,这可能包括一个访问计数,一个指向列表内部以方便线性遍历的运行指针,一个字符名,等等。当 T=1 时,信息包括 DLINK。

3)显然,列表是非常通用的结构;似乎完全可以说,当做出适当的约定时,无论什么样的结构都可表示为列表。由于列表的这种通用性,大量的程序设计系统都已被设计为便于进行列表的操作,而且通常在任何计算机装置当中都有好多个这样的系统可以利用。这样的系统是基于例如上面的(9)或(10)这样节点的通用格式,被设计成适合于列表操作的灵活性的。实际上,很显然,这种通用格式通常不是适合于特定应用的最好的格式,而且使用通用程序的处理时间显著地要比人们为特定的问题而"裁制"的系统的速度慢。例如,容易看出,迄今为止在本章中我们已经处理过的几乎所有的应用,都因采用(9)中或(10)中那样的通用的列表表示,而不是针对每种情况给出的节点格式,而受到损害。列表处理程序在处理节点时必须经常考察 T 字段,但在迄今我们的任何程序当中这并不需要。效率的这个损失在许多情况下由于程序设计相对容易和当使用通用系统时调试时间的减少而得到补偿。

4)在用于列表处理的算法和以前在本章中给出的算法之间也有极其重大的差别。由于列表可以被包含在许多其它的列表当中,因此这绝不意味着确切地清楚什么时候列表应该被送回到可利用的存储池中。每当 NODE(X)不再被需要时,迄今为止我们的算法总是说"AVAIL⇐X"。但由于一般的列表完全可以以不可预料的方式增长和消亡,因此通常很难确切地说什么时候特定的节点成为多余的了。因此用列表来维护可用空间表比起以前考虑的简单情况要困难得多。我们将在本小节的剩余部分专注于对存储回收问题的讨论。

让我们想像我们正设计一个将为数以百计的其他程序员所使用的通用列表处理系统。为维护可用空间表,这里提出两个主要的方法:使用**访问计数器**和**废料收集**。访问计数器技术在每个节点处利用一个新字段来包含指向这个节点指针的计数。当程序运

行时，维护这种计数是比较容易的，而一旦它变成零，问题中的节点就变成可用的了。另一方面，废料收集技术，在每一节点处要求一个称为标记位的新的一位字段。在这种情况下的想法是，把几乎所有的算法都编写成使得它们不把任何节点恢复成自由存储，而让程序快乐地运行，直到所有可用存储都用光为止；然后一个“回收”算法利用标记位来找出当前已不能访问的所有节点，并把它们恢复成可用存储。在这之后程序就可以继续运行了。

这两种方法都不完全令人满意。访问计数器方法的主要缺点是它并不总是释放可以利用的所有节点。它对于重叠的列表（即包含公共的子列表的列表）工作得很好；但对于递归的列表，像(4)中我们的例子 L 和 N，通过访问计数技术将绝不会恢复。即使当已无运行程序可以存取的其它列表指向它了，它们的计数也将是非零的（因为它们引用自身）。而且，访问计数方法在每个节点使用一大块空间（尽管由于计算机字的大小，这个空间有时是可用的）。

除了在每个节点中令人讨厌地损失一个二进位之外，废料收集技术的困难，在于当几乎所有内存空间都被使用时，它运行得非常缓慢；而且在这样的情况下，由回收过程找回的自由存储的数量不值得去作这个努力。超过存储容量的程序（许多未被调试的程序就是这样），通常浪费大量时间，在存储最终穷尽之前调用废料收集程序许多次，但几乎都是空无所获。对于这个问题的部分解决方法是让程序员指定一个数 k，指明当废料收集的运行已经找到 k 个或者小于 k 个自由节点时，就宣告该处理不应再继续进行下去。

另一个问题是有时难以精确地确定在给定的阶段里哪些列表不是废料。如果程序员已经使用任何非标准的技术或者在非通常的位置保持任何指针值，则废料收集程序出错的机会很大。在调试的历史中一些最大的谜是由于这样的事实引起的，即在以前已经工作了多次的程序运行期间，在出乎意料的时间里废料收集突然发生。废料收集也要求，程序员在所有时间里在所有指针字段里保留正确的信息，尽管我们经常发现，在程序不使用的字段——例如，在队列的末尾节点中的链接——保留无意义的信息是方便的；请参见习题 2.2.3-6。

尽管废料收集要求每个节点有一个标记位，但我们可以把所有标记位在另一个内存区域里组装在一起，并通过分开的表格以及通过节点的位置和它的标记位之间的适当对应加以保持。在某些计算机上，这个思想可能导致处理废料收集的另一种方法，此方法比起在每个节点处放弃一位来是更有吸引力的。

J. Weizenbaum 提议对访问计数技术做有趣的修改。使用双重链接列表结构，他仅在每个列表表头放置访问计数。因此，当指针变量遍历一个列表时，它们不被包括在对于个别节点的访问计数中。如果我们知道访问计数是对于整个列表来维护的这个规则，我们（在理论上）就知道怎样来避免访问其访问计数已成为零的任何列表。我们也有完全的自由确干脆不顾访问计数以及把特定的列表恢复成可用存储。这些思想要仔细对待；对于没有经验的程序员来说，已经证明它们是有些危险的，而且由于访问已被删去的节点的结果所致，而且它们趋向于使程序的调试更为困难。Weizenbaum 方法最漂亮的部分是他对于其访问计数刚刚变成零的列表的处理：把这种列表加在当前可用

表的末尾——对于双重链接列表这是容易做的——而且仅仅在所有以前可用的单元都被用完之后才考虑可用空间。最终，当这个列表的个别节点变成可用之后，这些节点访问的列表的访问计数器才减 1。相对于运行时间，被延迟的删除列表的动作是十分有效的；但它趋向于使不正确的程序正确地运行一会儿；关于进一步的细节，请见*CACM* **6** (1963)，524 ~ 544。

由于若干原因，废料收集的算法是十分值得关注的。首先，这样的算法，在我们要标记由给定节点直接或间接地访问的所有节点时，是有用的。（例如，像在习题 2.2.3-26 中那样，我们可能要找出为某个子程序直接或间接地调用的所有子程序。）

废料收集一般分两个阶段进行。我们假定，所有节点的标记位开始时都为零（或者我们把它们都置为零）。现在，第一阶段，从那些直接可被主程序访问的节点开始，标记所有非废料的节点。第二阶段对整个存储池区域进行一次顺序扫描，把所有未被标记的节点放进自由空间表中。标记阶段是最有趣的，因此我们将把注意力集中于它上面。然而，第二个阶段的某些变化使得它变成为不平凡的；见习题 9。

当一个废料收集算法运行时，仅有非常有限数量的存储可利用来控制标记过程。在下列的讨论中这个有趣的问题将变得很清楚；困难在于，当人们开始听到废料收集的想法时，大多数人都不欣赏它，因此过了许多年都没有好的解决方法。

下列的标记算法也许是最明显的。

算法 A（标记）　设供列表存储所使用的整个内存是 NODE(1)，NODE(2)，…，NODE(M)，并假设这些字或者是原子，或者包含两个链接字段 ALINK 和 BLINK。假定所有节点开始时都是未标记的。本算法的目的是标记在非原子节点中 ALINK 和/或 BLINK 指针的链可抵达的所有节点，而且从“直接可访问”的节点集合开始，即，由主程序中某些固定位置所指的节点开始；这些固定的指针被用作所有内存访问的来源。

A1. [初始化] 标记直接可访问的所有节点。置 K←1。

A2. [NODE(K)意味着另一个节点吗？] 置 K1←K + 1。如果 NODE(K)是原子或未标记，转到步骤 A3。否则，如果 NODE(ALINK(K))未标记：标记它，而且如果它不是原子，置 K1←min(K1，ALINK(K))。类似地，如果 NODE(BLINK(K))未标记：标记它，而且如果它不是原子，则置 K1←min(K1，BLINK(K))。

A3. [完成了吗？] 置 K←K1。如果 K≤M，转回步骤 A2；否则，算法结束。▌

在这个算法和这一小节后边的算法中，为方便起见，我们将假定，不存在的节点“NODE(Λ)”是已标记的。（例如，ALINK(K)或 BLINK(K)在步骤 A2 中可以等于 Λ。）

算法 A 的一个变形是在步骤 A1 中置 K1←M + 1，从步骤 A2 中删去操作“K1←K + 1”，而且把步骤 A3 改成

A3′. [完成了吗？] 置 K←K + 1。如果 K≤M，则返回步骤 A2。否则如果 K1≤M，置 K←K1 和 K1←M + 1 并且返回步骤 A2。否则算法结束。▌

给出对算法 A 的一个精确分析或者确定它比刚才给出的变形是更好些还是更坏些，是极其困难的。因为没有有意义的方法来描述输入出现的概率分布。我们可以说，

当 n 是它标记的单元数时,在最坏情况下,它花费同 nM 成正比的时间;而且一般说来,我们可以确信,当 n 很大时,它是非常缓慢的。算法 A 实在太慢了,以致不能使废料收集成为可用的技术。

另一种相当明显的标记算法,是沿着所有通路进行,而且当我们进行时,把分支点记录到栈上:

算法 B(标记) 利用 STACK[1],STACK[2],… 作为辅助存储来记住所有尚未被完成的通路,本算法实现和算法 A 相同的效果。

B1. [初始化] 令 T 是直接可访问的节点的个数;标记它们并且在 STACK[1],…,STACK[T]中放置指向它们的指针。

B2. [栈空了吗?] 如果 T = 0,则算法结束。

B3. [删去顶部条目] 置 K←STACK[T],T←T - 1。

B4. [考察链接] 若 NODE(K)是原子,则转回步骤 B2。否则,若 NODE(ALINK(K))是未标记的,则标记它而且置 T←T + 1,STACK[T]←ALINK(K);如果 NODE(BLINK(K))是未标记的,标记之并置 T←T + 1 ,STACK[T]←BLINK(K),转回B2。 ▌

算法 B 显然有实质上同它标记的单元数成比例的执行时间,而且这和我们可以预料的一样好;但是它对于废料收集实际上也是不能用的,因为没有地方来保存栈!假定在算法 B 中的栈可以增长到比如说内存大小的百分之五,这并非不合理;但当调用废料收集程序时,所有可用空间已经用光了,只有固定数目(稍稍小的数目)的单元可用作这样的栈。大多数早期的废料收集过程实质上都是基于这个算法。如果特殊的栈空间被用光,则整个算法就需结束。

使用固定的栈大小,把算法 A 和算法 B 组合起来,有可能有较好的选择。

算法 C(标记) 使用 H 个单元的一个辅助表格 STACK[0],STACK[1],…,STACK[H - 1],这个算法实现与算法 A 和 B 同样的效果。

在这个算法中,动作"把 X 插入栈中"意义如下:"置 T←(T + 1)mod H 和 STACK[T]←X。如果 T = B,则置 B←(B + 1)mod H 以及 K1←min(K1,STACK[B])。"(注意 T 指向栈当前的顶部,而且 B 指向当前底部之下的一个位置;STACK 实质上作为一个输入受限双端队列一样操作。)

C1. [初始化] 置 T←H - 1,B←H - 1,K1←M + 1。标记所有直接可访问的节点,并且逐次地把它们的单元插入栈中(如同上面刚描述的那样)。

C2. [栈空了吗 ?] 如果 T = B,转到 C5。

C3. [删去顶部条目] 置 K←STACK[T],T←(T - 1) mod H。

C4. [考察链接] 如果 NODE(K)是一个原子,返回步骤 C2。否则,如果 NODE(ALINK(K))是未标记的,则标记它并把 ALINK(K)插入栈上。类似地,如果 NODE(BLINK(K))是未标记的,则将其标记且将 BLINK(K)插入栈。返回 C2。

C5. [清除] 如果 K1 > M,则算法结束。(变量 K1 代表最小位置,该处存在指向应该标记的节点的新前置量的可能性。否则,如果 NODE(K1)是原子或是未标记的,则 K1 增 1 并且重复本步骤。如果 NODE(K1)是已标记的,则置 K←K1,K1 增 1,而且转到 C4。 ▌

当 NODE(X)是原子时,如果 X 从不被放入栈上,则本算法和算法 B 可加以改进;而且,当步骤 B4 和 C4 知道一些项将被立即删除时,它们不必把这些项放到栈上。这样的修改是直截了当的,因此它们已被省去以避免使这些算法不必要地复杂化。

当 H = 1 时,算法 C 实质上等价于算法 A,而当 H = M 时,等价于算法 B。当 H 变得更大时,它逐渐地变得更有效。不幸,算法 C 由于和算法 A 同样的原因,不能进行精确的分析,而且我们对于多大的 H 将使这个算法足够地快心里没底。在大多数应用中,像 H = 50 这样一个值足以使算法 C 可用作废料收集,这样说似乎是对的,但是却不令人满意。

算法 B 和 C 使用保持在顺序的内存单元中的一个栈;但是在本章早些时候已经看到,链接存储技术很好地适合于保持在内存中不连续的栈。这就提示了一个想法,即我们可以使算法 B 的栈散列于我们正在进行废料收集的同一个内存区域中。如果我们为废料收集提供多一点供其喘气的空间,则这可以很容易地完成。例如,我们假定,所有列表被表示成像(9)中那样,只是列表头节点的 REF 字段被用作废料收集的目的而不是作为访问计数。那么我们可以重新设计算法 B 使得栈被维持在表头节点的 REF 字段中。

算法 D(标记)　这个算法实现和算法 A,B 及 C 同样的效果,但是如上面所描述那样,它假定节点有 S,T,REF 和 RLINK 字段,而不是 ALINK 和 BLINK。S 字段被用做标记位,使得 S(P) = 1 意味着 NODE(P)是已标记了的。

D1. [初始化] 置 TOP←Λ。然后对于一个直接可访问列表的每一个指针 P(见算法 A 的步骤 A1),如果 S(P) = 0,则置 S(P)←1,REF(P)←TOP,TOP←P。

D2. [栈为空吗?] 如果 TOP = Λ,则算法结束。

D3. [删去栈顶条目] 置 P←TOP,TOP←REF(P)。

D4. [在整个列表中移动] 置 P←RLINK(P);然后如果 P = Λ,或者如果 T(P) = 0,转到 D2。否则置 S(P)←1。如果 T(P) > 1,则置 S(REF(P))←1(由此标记原子信息),否则(T(P) = 1),置 Q←REF(P);如果 Q ≠ Λ,且 S(Q) = 0,则置 S(Q)←1,REF(Q)←TOP,TOP←Q。重复步骤 D4。 ▌

可以对照算法 D 和算法 B,它们十分类似,而且运行时间实质上同被标记的节点个数成比例。然而,如果未经定性分析我们不能推荐算法 D,因为看上去有点温和的限制对于一般的列表处理系统来说经常是太严格了。这个算法实质上要求,像在(7)中一样,每当废料收集投入行动时,所有列表结构都是明确地形成了的。但是列表操作的算法暂时地都保持列表结构为畸形的,因此在那些短暂的期间里,像算法 D 这样的废料收集程序不得使用。而且当程序包含列表中点的指针时,在进行步骤 D1 时还必须小

心。

这些考虑把我们带到算法 E,它是由 Peter Deutsch 及由 Herbert Schorr 和 W. M. Waite 于 1965 年分别独立地发现的优美的标记算法。用于这个算法的假定只是稍稍地不同于算法 A 到算法 D。

算法 E(标记) 假定给定节点的集合,且有下列字段:

MARK (一位字段)

ATOM (另一个一位字段)

ALINK (指针字段)

BLINK (指针字段)

当 ATOM = 0 时,ALINK 和 BLINK 字段可以包含 Λ 或指向相同格式的另一节点的指针;当 ATOM = 1 时,ALINK 和 BLINK 字段的内容同本算法无关。

给定非空指针 P0,这个算法置在 NODE(P0)中的 MARK 字段为 1,并且置其它节点中,从 NODE(P0)通过在具有 ATOM = MARK = 0 的节点中的 ALINK 和 BLINK 指针链可达到节点的 MARK 字段也为 1。这个算法使用三个指针变量 T,Q 和 P。它以这样一个方式来修改链接和控制位,即在完成之后,所有的 ATOM,ALINK 和 BLINK 字段都被恢复成原来的设定,尽管它们可能暂时地被改动。

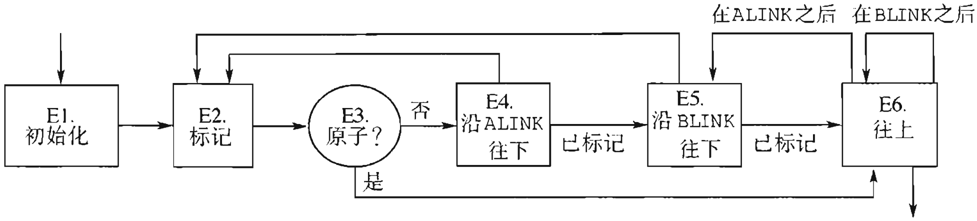


图 38 无辅助栈空间的用于标记的算法 E

E1. [初始化] 置 T←Λ,P←P0。(在本算法的剩余部分,变量 T 有双重的意义:当 T≠Λ 时,它像在算法 D 中一样,指向实质上是栈的顶部元素;而 T 所指向的节点曾经包含等于 P 的一个链接,代替当前占有 NODE(T)的"人工的"栈链接。

E2. [标记] 置 MARK(P)←1。

E3. [原子?] 如果 ATOM(P) = 1,转到 E6。

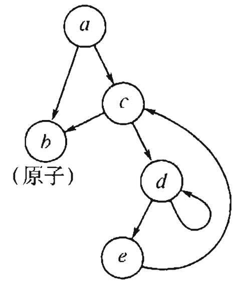
E4. [沿 ALINK 往下] 置 Q←ALINK(P)。如果 Q≠Λ,而且 MARK(Q) = 0,则置 ATOM(P)←1,ALINK(P)←T,T←P,P←Q 并转到 E2。(这里 ATOM 字段和 ALINK 字段暂时被修改,使得在某些已标记节点中的列表结构相当大幅度地被改变,但这些变化在步骤 E6 中将被恢复。)

E5. [沿 BLINK 往下] 置 Q←BLINK(P)。如果 Q≠Λ,而且 MARK(Q) = 0,则置 BLINK(P)←T,T←P,P←Q,并转到 E2。

E6. [往上] (这一步骤恢复在步骤 E4 和 E5 中所作的链接转换;ATOM(T)的设定告知是否要恢复 ALINK(T)或 BLINK(T)。)如果 T = Λ,算法结束。否则置 Q←T。如果 ATOM(Q) = 1,则置 ATOM(Q)←0,T←ALINK(Q),ALINK(Q)←P,P←Q,并返

回 E5。若 ATOM(Q) = 0，则置 T←BLINK(Q)，BLINK(Q)←P，P←Q，并重复 E6。▌

运行中的这个算法的例子出现于图 39 中，它示出对于简单的列表结构遇到的每一步。读者将发现非常仔细研究算法 E 是值得的；注意在步骤 E4 和 E5 中链接结构是怎样被人为地改变，以便保持类似于算法 D 中的栈的。当我们返回以前的状态时，ATOM 字段被用来说明 ALINK 或 BLINK 是否包含人为的地址。在图 39 底部所示的的“嵌套”说明在算法 E 执行期间每个非原子的节点如何被访问三次；在步骤 E2，E5 和 E6 的开始处出现相同的格局(T，P)。



a	ALINK[MARK]	b[0]	.	Λ[1]	.	b	.	.	.	.	.	.	.	.	.	.
	BLINK[ATOM]	c[0]	.	[1]	.	[0]	Λ	.	.	.	.	.	.	.	.	c
b	ALINK[MARK]	–[0]	.	.	[1]	.	.	.	.	.	.	.	.	.	.	.
	BLINK[ATOM]	–[1]	.	.	.	.	.	.	.	.	.	.	.	.	.	.
c	ALINK[MARK]	b[0]	.	.	.	.	.	[1]	.	.	.	.	.	.	.	.
	BLINK[ATOM]	d[0]	.	.	.	.	.	.	a	.	.	.	.	.	d	.
d	ALINK[MARK]	e[0]	.	.	.	.	.	.	.	c[1]	.	.	e	.	.	.
	BLINK[ATOM]	d[0]	.	.	.	.	.	.	.	[1]	.	.	[0]	.	.	.
e	ALINK[MARK]	Λ[0]	.	.	.	.	.	.	.	.	[1]	.	.	.	.	.
	BLINK[ATOM]	c[0]	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	T —		Λ	a	a	Λ	a	a	c	d	d	d	c	c	a	Λ
	P —		a	b	b	a	c	c	d	e	e	e	d	d	c	a
	下一步	E1	E2	E2	E6	E5	E2	E5	E2	E2	E5	E6	E5	E6	E6	E6

嵌套

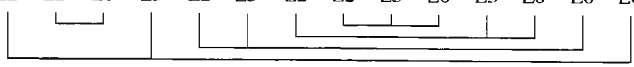


图 39　由算法 E 标记的结构
（这个表格仅示出自上一步以来已经出现的变化）

算法 E 的有效性可以通过使用对被标记节点个数的归纳法来证明。同时我们证明，在算法结束时，P 返回它的初始值 P0；其细节请见习题 3。如果删去步骤 E3 而且把对“ATOM(Q) = 1”的测试及适当的动作放在步骤 E4 和 E5 当中，把对“ATOM(P0) = 1”的测试放在步骤 E1 当中，则算法 E 将运行得更快。为简单起见我们以现有形式叙述了算

法;刚才指出的修改见于习题 4 的答案中。

用于算法 E 的思想可以应用于废料收集以外的问题;事实上,在习题 2.3.1-21 中已经提到它对树遍历的应用。读者还将发现,把算法 E 同在习题 2.2.3-7 中求解的较简单问题作一比较,是有用的。

在我们已经讨论的所有标记算法当中,仅有算法 D 是直接可应用于像在(9)中那样表示的列表的。其它算法都要测试给定的节点是否为原子,而(9)的约定是同这样的测试不相容的,因为除标记位外它们允许把原子信息填入整个字中。然而,其它每个算法都可被修改,使得当原子数据可以和字中的指针数据相区别时,它们就将有效,其中的指针数据是链接到字上的而不是着眼于字本身。在算法 A 或 C 中,我们可以简单地避免标记原子字直到把所有非原子的字段都适当地标记为止;然后再加上对于所有数据的另一遍扫描,就足以标记所有原子字了。算法 B 甚至更容易修改,因为我们仅须把原子字保留在栈外。对算法 E 的修改几乎也同样简单,尽管如果允许 ALINK 和 BLINK 都指向原子数据,它将有必要在非原子节点中引进另一个二进位的字段。这一般来说不难做到。(例如,当每节点有两个字时,每个链接字段的最低有效位可被用来存储临时信息。)

尽管算法 E 要求同它标记的节点数成比例的时间,但这个比例常数不像在算法 B 中的那么小;如同习题 5 中所讨论的那样,已知的最快的废料收集方法把算法 B 和 E 组合在一起。

现在让我们尝试对废料收集的效率作某些定量的估计,这同本章中用于以前的大多数例子的“ALINK⇐X”的原理相反。在以前的每种情况下,我们可能省略了对把节点返回给自由空间的特别提及,而代之以提及被替入的废料收集程序。(和一组通用的列表操作子程序相反,在专用的应用中,废料收集程序的程序设计和调试是比我们使用过的方法要更加困难的。而且,当然,废料收集要求在每个节点中保留额外的一个二进位;但是在这里我们感兴趣的是一旦程序已经被写成和调试时,它们的相对速度。)

已知的最好的废料收集程序实质上有形如 $c_1\mathrm{N}+c_2\mathrm{M}$ 的执行时间,其中 c_1 和 c_2 是常数,N 是被标记的节点数,而 M 是内存中总的节点数。因此,M − N 是被找到的自由节点数,为把这些节点返回到自由存储,所需要的时间数量是每个节点为 $(c_1\mathrm{N}+c_2\mathrm{M})/(\mathrm{M}-\mathrm{N})$。令 $\mathrm{N}=\rho\mathrm{M}$;这个数就成为 $(c_1\rho+c_2)/(1-\rho)$。所以如果 $\rho=3/4$,即如果内存是四分之三满时,则把每个自由节点返回到存储中要花费 $3c_1+4c_2$ 单位时间。当 $\rho=\dfrac{1}{4}$ 时,相应的花费只有 $\dfrac{1}{3}c_1+\dfrac{4}{3}c_2$。如果我们不使用废料收集技术,把每个节点返回的时间数量实质上是一个常数 c_3,因而 c_3/c_1 是否将很大是值得怀疑的。因此可以看出,当内存变满时,在什么程度上废料收集是低效的,而当对内存的要求是低的时,它如何相应地有效。

许多程序都有这样一种性质,即好节点同总内存量之比 $\rho=\mathrm{N}/\mathrm{M}$ 是十分小的。在这样的情况下当内存池变满的时候,可能最好是使用复制技术(见习题 10),而不必操心保持复制的节点的内容,以便把所有活动的列表数据的内容移到同样大小的另一个内存池中。然后当第二个内存池满了时,我们可以再把数据移到前一个内存池中。通

过这个方法,就可以同时在高速内存中保存更多的数据,因为链接字段倾向于指向临近的节点。而且,不需要有标记阶段,因而存储分配就只是顺序的了。

有可能把废料收集同把单元返回到自由存储的其它方法组合在一起;这些想法不是互相排斥的,因而某些系统除了允许程序员明确地抹去节点之外,还同时使用访问计数器和废料收集方案。其想法是每当返回单元的所有其它方法都失灵时,使用废料收集作为“最后的求助手段”。L. P. Deatsch 和 D. G. Bobrow 在*CACM* **19** (1976),522 ~ 526 中描述了精心制作的系统,它实现了这个想法并且还包括搁置关于访问计数的操作的机制以便获得更高效率。

也有可能使用列表的顺序表示,而以更复杂的存储管理为代价,它可节省许多链接字段。请参见 N.E. Wiseman 及 J.O. Hiles,*Comp. J.* **10** (1968), 338 ~ 343; W. J. Hansen, *CACM* **12** (1969),499 ~ 507;以及 C.J. Cheney, *CACM* **13** (1970),677 ~ 678。

Daniel P. Friedman 和 David S. Wise 已经发现,在许多情况下,即使当列表指向它们自己时,如果不把某些链接字段包括在计数中,仍可相当令人满意地应用访问计数器方法[*Inf. Proc. Letters* **8** (1979),41 ~ 45]。已经提出了大量的废料收集算法的变形和改进。Jacques Cohen 在*Computing Surveys* **13** (1981),341 ~ 367 中,对于 1981 年以前的文献给出了详尽的评述,并且对于数据页在慢速存储器和快速存储器之间来回穿梭时内存访问的额外代价问题作了重要的评述。

如同我们已经对它所作的描述那样,废料收集不适合于“实时”应用,因为在那里每个基本的列表操作都必须很快;即使废料收集不大经常被调用,在那样一些机会它也要求大量的计算机时间,习题 12 讨论了实时的废料收集成为可能的某些方法。

It is a very sad thing nowadays
that there is so little useless information.
现今一件非常令人忧虑的事情是
无用的信息竟这么少。
——OSCAR WILDE (1894)

习 题

▶ 1.[*M21*] 在 2.3.4 小节我们看到树是有向图的“经典”数学概念的特殊情况。能否以图论的术语描述列表?

2.[*20*] 在 2.3.1 小节我们看到在计算机内用穿线表示可以方便树的遍历,能否以类似的方式对列表结构进行穿线?

3.[*M26*] 证明算法 E 的正确性。[提示:参见算法 2.3.1 T 的证明。]

4.[*28*] 假定把节点表示为一个 MIX 字,且 MARK 占有(0:0)字段[“+” = 0,“-” = 1],ATOM 占有(1:1)字段,ALINK 占有(2:3)字段,BLINK 占有(4:5)字段,以及 Λ = 0,试编写算法 E 的 MIX 程序。借助于相关参数,确定你的程序的执行时间。(在 MIX 计算机内,确定内存单元是否包含 -0 或 +0 的问题并非不足道的,而这可能成为你的程序的一个方面。)

5.[*25*] (Schorr 和 Waite) 试给出一个标记算法,它组合算法 B 和 E 如下:保留算法 E 关于在节点内诸字段等的假定;但是像在算法 B 中那样使用辅助栈 STACK[1],STACK[2],…,STACK[N],而且仅当栈满时才使用算法 E 的机制。

6.[*00*] 在本节末尾处关于定量的讨论指出废料收集的开销近似为 $c_1\mathrm{N}+c_2\mathrm{M}$ 个时间单位;"$c_2\mathrm{M}$"项来自何处?

7.[*24*] (R.W.Floyd) 试设计标记算法,在不使用辅助栈这点上类似于算法 E,但有两点例外:(i)它有更困难的任务要做,因为每个节点仅包含 MARK,ALINK 和 BLINK 字段——没有 ATOM 字段来提供附加的控制;不过(ii)它也有较简单的任务要做,因为它只标记一个二叉树而不是一般的列表。这里 ALINK 和 BLINK 是在二叉树中通常的 LLINK 和 RLINK。

▶ 8.[*27*](L.P.Deutsch) 试设计在不使用辅助存储作为栈这点上类似于算法 D 和算法 E 的标记算法,但修改其方法,使得它对于有可变大小的节点和个数变化的且有下列格式的指针仍有效:节点的第一个字有两个字段 MARK 和 SIZE;对 MARK 字段如同在算法 E 中那样处理,而 SIZE 字段含有一个数 $n\geq 0$。这意味着在第一个字之后有 n 个连续的字,每一个都含两个字段 MARK(它为零而且应保持不变)及 LLINK(它为 Λ 或指向另一个节点的第一个字)。例如,有三个指针的一个节点将由四个连续的字组成:

第一个字	MARK = 0(将被置成 1)	SIZE = 3
第二个字	MARK = 0	LLINK = 第一个指针
第三个字	MARK = 0	LLINK = 第二个指针
第四个字	MARK = 0	LLINK = 第三个指针

你的算法应当对于从给定节点 P0 出发可抵达的所有的点都作标记。

▶ 9.[*28*] (D.Edwards) 试设计废料收集第二阶段的算法,它在下列意义下使"存储紧凑":令 NODE(1),…,NODE(M)是算法 E 中描述的具有 MARK,ATOM,ALINK 及 BLINK 字段的一个字的节点。假定在所有不是废料的节点中 MARK = 1。所要求的算法在必要时应使被标记的节点浮动,使得所有这些节点出现在连续的单元 NODE(1),…,NODE(K)中,同时非原子节点的 ALINK 和 BLINK 字段必要时应予修改使得列表结构得以保持。

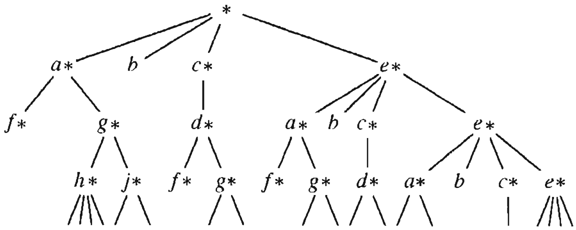
▶ 10.[*28*] 假定使用像(7)中那样的内部表示,试设计复制列表结构的算法。(因此,如果要求你的过程复制其表头是在(7)的左上角处的节点的列表,则应建立有 14 个节点的新列表集合,且其结构和信息等于(7)中所示的结构和信息。)

假定像在(9)中一样使用 S,T,REF 和 RLINK 字段把列表结构存在内存中,而且 NODE(P0)是要被复制的列表的表头。进一步假定在每一个列表表头节点中的 REF 字段为 Λ;为避免对于附加的内存空间的需要,你的复制过程应当利用 REF 字段(而且之后应再把它们复原成 Λ)。

11.[*M30*] 通过重复所有重叠的元素直到不再存在为止,任何列表结构都可被"充分地扩展"成为一个树结构;当列表是递归的时,这给出一个无穷树。例如,列表(5)将被扩展成其头四层如下页所示的无穷树,试设计一个算法检测两个列表结构的等价性,这种等价性的意义是,当充分扩展时它们有相同的框图。例如,如果

$$A=(a:C,b,a:(b:D))$$
$$B=(a:(b:D),b,a:E)$$
$$C=(b:(a:C))$$
$$D=(a:(b:D))$$
$$E=(b:(a:C))$$

则在这个意义下列表 A 和 B 等价。



12.[30] (M. Minsky) 证明:当计算机控制某个物理装置时,甚至当对于执行的每个列表操作所要求的极大执行时间设置严格的上限时,仍有可能在“实时”应用中可靠地使用废料收集方法。[提示:如果相当小心,可以把废料收集安排成并行于列表操作。]

2.4 多重链接结构

既然我们已经详细地考察了线性表和树结构,在计算机内表示结构信息的原理应当是明显的了。在本节中,我们将考察这些技术的另一种应用,这次是对于信息结构稍微更复杂的典型情况进行的:在高级应用中,通常好多种类型的结构同时出现。

"多重链接结构"涉及在每个节点中有若干个链接字段的那些节点,而不是像在大多数我们以前的例子中那样仅仅涉及一两个链接而已。我们已经看到过多重链接的一些例子,例如在2.2.5小节中的模拟电梯系统以及在2.3.3小节中的多个变量的多项式那样。

我们将看到,在每个节点中许多个不同类型的链接的存在,并不必然地使伴随的算法比起已经讨论过的算法更难写或更难理解。我们还将讨论"*有多少结构信息应当明确地记录在内存中*"这样一个重要的问题。

我们将要考虑的这个问题是在同编写用于翻译COBOL及相关语言的编译程序相关联当中产生的。使用COBOL的程序员可能在好多层次上对程序变量赋予字符名字;例如,程序可能访问有下列结构的销售和购买的数据文件:

```
1 SALES                    1 PURCHASES
  2 DATE                     2 DATE
    3 MONTH                    3 DAY
    3 DAY                      3 MONTH
    3 YEAR                     3 YEAR
  2 TRANSACTION              2 TRANSACTION                  (1)
    3 ITEM                     3 ITEM
    3 QUANTITY                 3 QUANTITY
    3 PRICE                    3 PRICE
    3 TAX                      3 TAX
    3 BUYER                    3 SHIPPER
      4 NAME                     4 NAME
      4 ADDRESS                  4 ADDRESS
```

这个格局指出在SALES中的每个项由两个部分组成,即DATE和TRANSACTION;DATE被进一步分成三个部分,而TRANSASCTION同样有五个部分。类似的说明适用于PURCHASES。这些名字的相对顺序指出这些量在文件的外部表示(例如磁带或打印的形式)中的顺序;注意在这个例子里,"DAY"和"MONTH"在两个文件中以相反顺序出现。程序员也给出未在这个图解中示出的信息,告知每项信息占据多大的空间以及以什么格式出现;这样的考虑在这一节里同我们无关,所以我们将不进一步去提及。

COBOL程序员首先描述文件的布局以及其它程序变量,然后确定对这些量进行操作的算法。为了访问在上面的例子中的个别变量,仅仅给出名字DAY是不充分的,因为无法告知称为DAY的变量是在SALES文件中或是在PURCHASES文件中。因此给予

COBOL 程序员写"DAY OF SALES"的能力来指 SALES 中的 DAY 部分。程序员也可以写得更完备些。例如：

"DAY OF DATE OF SALES"

但是一般来说不必给出比避免二义性所必需的更多的限定。因此，

"NAME OF SHIPPER OF TRANSACTION OF PURCHASES"

可以缩写成

"NAME OF SHIPPER"

因为只有一个部分的数据称做 SHIPPER。

COBOL 的这些规则可以更精确地叙述如下：

a)每个名称紧前头都有称为它的*层次号*的关联正整数。名称或者指*初等项*，或者它是名称所跟着的一个或多个项的组的名称。在后一种情况下，这个组的每项必有相同的层次号。(例如，上面的 DATE 和 TRANSACTION 有层次号 2，它大于 SALES 的层次号 1。)

b)为访问名为 A_0 的一个初等项或项的组，一般形式为

$$A_0 \text{ OF } A_1 \text{ OF } \cdots \text{ OF } A_n$$

其中 $n \geq 0$，而且对于 $0 \leq j \leq n$，A_j 是直接或间接地在称为 A_{j+1} 的组之内的某项的名称。必定有恰好一项 A_0 满足这个条件。

c)如果在若干个位置出现同一个名字 A_0，则必须有办法通过条件限制访问其中的每一个。

作为规则 c)的例子，数据配置

```
1  AA
   2  BB
      3  CC                (2)
      3  DD
   2  CC
```

将不被允许，因为没有访问 CC 的第二个出现的无二义性的方法(见习题 4)。

COBOL 有另一个特征，它影响编译程序的编写和我们正在考虑的应用，这就是在该语言中的一个选项，它使得有可能同时访问许多项。COBOL 程序员可以写出

MOVE CORRESPONDING α TO β

把具有相应名字的所有项从数据区域 α 移动到数据区域 β 去。例如，COBOL 语句

MOVE CORRESPONDING DATE OF SALES TO DATE OF PURCHASES

将意味着来自 SALES 文件的 MONTH，DAY 和 YEAR 的值要被移动到 PURCHASES 文件的 MONTH，DAY 和 YEAR 中去。(DAY 和 MONTH 的相应顺序因此被交换。)

在本节中我们将研究的问题是设计适合于在 COBOL 的编译程序中使用的三个算法，它们是来做下列的事情的：

操作 1　处理像(1)中那样名字和层次号的描述,应把相关信息放入编译程序内的表格中供在操作 2 和操作 3 中使用。

操作 2　确定如规则 b)中那样的给定的合格访问是否有效,而当有效时,要找出对应的数据项的位置。

操作 3　找出由给定的 CORRESPONDING 语句指出的所有对应的项目对。

我们将假定编译程序有“符号表子程序”,它把字符名转换成指向该名的表条目的链接。(第 6 章中详细讨论了用于构造符号表的算法。)除了符号表之外还有一个更大的表,其中包含要编译的 COBOL 源程序中的每个数据项的一个条目;我们将称之为数据表。

显然,在我们知道在数据表中要存的是什么类型的信息之前,不可能设计操作 1 的算法,而且数据表的形式依赖于我们需要什么信息以便实现操作 2 和 3;因此首先考察操作 2 和 3。

为了确定 COBOL 访问

$$A_0 \text{ OF } A_1 \text{ OF } \cdots \text{ OF } A_n, \quad n \geq 0 \tag{3}$$

的意义,应当首先在符号表中查 A_0 这个名字。对于这个名字应当有从符号表条目到所有数据表条目的一系列链接。然后对于每个数据表条目,我们需要有一个对于包含该条目的组项的条目的链接。现在如果有从数据表项回到符号表的进一步的链接字段,不难看出可以如何处理像(3)一样的访问。其次,我们将需要从对于组项的数据表条目到在这组中的诸项的某种类型的链接,以便放置由“MOVE CORRESPONDING”指出的对偶。

因此我们在每个数据表条目中发现对于五个链接字段潜在的需要:

PREV(对于具有相同的名字的以前条目的链接,如果有的话);

PARENT(对于包含这个项的最小组的链接,如果有的话);

NAME(对于这个项的符号表条目的链接);

CHILD(对于组的第一个子项的链接);

SIB(对包含这个项的组中的下一个子项的链接)。

显然,像上面那样的对于 SALES 和 PURCHASES 的 COBOL 数据结构实质上是树;而这里出现的 PARENT,CHILD 和 SIB 链接从我们以前的研究看,是熟悉的。(一棵树的传统的二叉树表示由 CHILD 和 SIB 链接组成;加上 PARENT 链接给了我们所谓的“三重链接树”。上面的五个链接由这三个链接连同 PREV 和 NAME 组成,后两个向树结构添加进一步的信息。)

或许并非所有这五个链接都会是必要的或充分的,但我们将首先尝试在试验性的假设,即数据表条目将涉及这五个链接字段(加上与我们的问题无关的进一步的信息)之下设计我们的算法。作为使用多重链接的例子,考虑(4)所示的两个 COBOL 数据结构,它们将像在(5)中所示那样加以表示(连同以符号指出的链接)。每一个符号表条目的 LINK 字段指向问题中符号名最近遇到的数据表条目。

```
1  A                1 H
   3  B                5  F
      7  C                8  G
      7  D             5  B                         (4)
   3  E                5  C
   3  F                   9  E
      4  G                9  D
                          9  G
```

我们需要的第一个算法是以这样一种形式构造数据表的算法。注意在 COBOL 规则中允许的选择层次号码的灵活性;(4)左边的结构完全等价于

```
1  A
   2  B
      3  C
      3  D
   2  E
   2  F
      3  G
```

因为层次号码不必是顺序的。

符号表

		LINK
A:		A1
B:		B5
C:		C5
D:		D9
E:		E9
F:		F5
G:		G9
H:		H1

空框指明附加信息与此处无关

数据表

	PREV	PARENT	NAME	CHILD	SIB	
A1:	Λ	Λ	A	B3	H1	
B3:	Λ	A1	B	C7	E3	
C7:	Λ	B3	C	Λ	D7	
D7:	Λ	B3	D	Λ	Λ	
E3:	Λ	A1	E	Λ	F3	
F3:	Λ	A1	F	G4	Λ	
G4:	Λ	F3	G	Λ	Λ	
H1:	Λ	Λ	H	F5	Λ	
F5:	F3	H1	F	G8	B5	
G8:	G4	F5	G	Λ	Λ	
B5:	B3	H1	B	Λ	C5	
C5:	C7	H1	C	E9	Λ	
E9:	E3	C5	E	Λ	D9	
D9:	D7	C5	D	Λ	G9	
G9:	G8	C5	G	Λ	Λ	

(5)

然而，某些层次号的顺序是非法的；例如，如果(4)中 D 的层次号被改成“6”(在任何一个位置)，我们就将有无意义的数据配置，违背了一组中的所有项都必须有相同号码的规则。因此下列算法确保 COBOL 的规则 a)不被破坏。

算法 A(构造数据表) 这个算法给出对偶序列(L，P)，其中 L 是正整数的“层次号”，而 P 指向对应于如上的(4)那样的 COBOL 数据结构符号表条目。像在上面的例(5)那样，这个算法构造一个数据表。当 P 指向以前还未出现的符号表条目时，LINK(P)将等于 Λ。这个算法使用和通常一样处理的辅助栈(或者像在 2.2.2 小节一样，使用顺序存储单元，或者像在 2.2.3 小节一样，使用链接分配)。

A1. [初始化] 置栈的内容为单个条目(0，Λ)。(整个算法中栈的条目是对偶(L，P)，其中 L 是整数而 P 是指针。当算法进行时，栈包含层次号以及对比当前层号高的所有层上最近的数据条目的指针。例如，在上面的例子中，在遇到对偶“3，F”之前，从栈底到栈顶，栈将包含

$$(0,\Lambda)\quad(1,\mathrm{A1})\quad(3,\mathrm{E3})。)$$

A2. [下一项] 设(L，P)是来自输入的下一数据项。然而，如果输入被穷尽，则算法结束。置 Q⇐AVAIL(即，令 Q 是可以放置下一个数据条目的新节点的单元)。

A3. [置名字链接] 置 PREV(Q)←LINK(P)，LINK(P)←Q，NAME(Q)←P。(这适当地设置 NODE(Q)中的五个链接的两个。我们现在要适当地设置 PARENT，CHILD 和 SIB。)

A4. [比较层次] 设栈的顶部条目是(L1，P1)。如果 L1 < L，则置 CHILD(P1)←Q(或者如果 P1 = Λ，则置 FIRST←Q，其中 FIRST 是将指向第二个数据条目的变量)并转到 A6 去。

A5. [删除顶部层] 如果 L1 > L，删去顶部栈条目，令(L1，P1)是刚进到栈顶的新条目，并重复步骤 A5。如果 L1 < L，则宣告出错(在同一层次上出现了混合号码)。否则，即当 L1 = L 时，置 SIB(P1)←Q，删去栈顶条目，并置(L1，P1)为刚刚进入栈顶的对偶。

A6. [置家族链接] 置 PARENT(Q)←P1，CHILD(Q)←Λ，SIB(Q)←Λ。

A7. [添加到栈上] 在栈的顶部放置(L，Q)，并返回 A2。 ▍

如同在步骤 A1 中说明的那样，引入辅助栈使这个算法变成如此明显，因而无需进一步的说明。

下一个问题是找出对应于访问

$$A_0 \text{ OF } A_1 \text{ OF } \cdots \text{ OF } A_n, \qquad n\geqslant 0 \tag{6}$$

的数据表条目的位置。好的编译程序也将检查以确保这种访问是无二义性的。在这种情况下，一个适当的算法立即就提了出来：我们所要做的是对于名字 A_0 跑遍整个的数据表条目的清单，并确保恰有这些条目之一匹配所述的条件的 $A_1,\cdots,A_n$。

算法 B(检查合格的访问) 对应于访问(6)，符号表子程序将分别地寻找对于 A_0，$A_1,\cdots,A_n$ 的指向符号表条目的指针 $P_0,P_1,\cdots,P_n$。

本算法的目的是检查 $P_0, P_1, \cdots, P_n$,或者确定访问(6)有错,或者把变量 Q 置成由(6)所访问的数据表条目的地址。

B1.[初始化] 置 Q←Λ,P←LINK(P_0)。

B2.[完成了吗?] 如果 P = Λ,则算法结束;这时如果(6)不对应于任何数据表条目,则 Q 将等于 Λ。但若 P≠Λ,置 S←P 和 k←0。(S 是将从 P 开始通过 PARENT 链接往树上运行的一个指针变量;k 是从 0 到 n 的整型变量。实际上,指针 $P_0, \cdots, P_n$ 通常将被保持在一个链接表中,而且不使用 k,我们将代之以使用遍历这个表的一个指针变量,见习题 5。)

B3.[匹配完了?] 如果 $k < n$ 转到 B4。否则我们已找到匹配的数据表条目;如果 Q≠Λ,这是找到的第二个条目,因此指明出错条件。置 Q←P,P←PREV(P),并转到 B2。

B4.[k 增 1] 置 k←$k+1$。

B5.[沿树上移] 置 S←PARENT(S)。若 S = Λ,我们没能找到匹配;置 P←PREV(P)并转到 B2。

B6.[A_k 匹配吗?] 如果 NAME(S) = P_k,则转到 B3;否则转到 B5。 ▌

注意在这个算法中不需要 CHILD 和 SIB 链接。

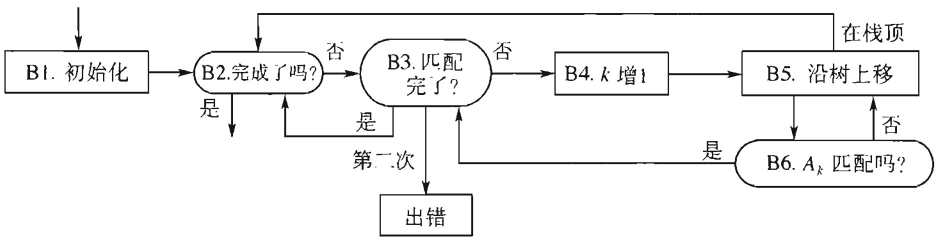


图 40　校验一个 COBOL 访问的算法

我们需要的第三和最后一个算法涉及“MOVE CORRESPONDING”;在我们设计这样的算法之前,必须对需要什么作精确定义。COBOL 语句

$$\text{MOVE CORRESPONDING } \alpha \text{ TO } \beta \tag{7}$$

(其中 α 和 β 是像(6)一样的对数据项的访问),是对所有语句集合

$$\text{MOVE } \alpha' \text{ TO } \beta'$$

的缩写,其中存在整数 $n \geqslant 0$ 以及 n 个名字 $A_0, A_1, \cdots, A_{n-1}$,使得

$$\begin{aligned} \alpha' &= A_0 \text{ OF } A_1 \text{ OF } \cdots \text{OF } A_{n-1} \text{ OF } \alpha \\ \beta' &= A_0 \text{ OF } A_1 \text{ OF} \cdots \text{OF } A_{n-1} \text{ OF } \beta \end{aligned} \tag{8}$$

而且 α' 或者 β' 是初等项(而非组项)。其次,我们要求(8)的第一层示出完备限定,即对于 $0 \leqslant j < n$,A_{j+1} 是 A_j 的父亲;α' 和 β' 在这种树中必须恰好要比 α 和 β 低 n 层。

相对于我们的例子(4),

MOVE CORRESPONDING A TO H

因此是语句

MOVE B OF A TO B OF H

MOVE G OF F OF A TO G OF F OF H

的缩写。

这个识别所有对应的对偶 α',β' 的算法是十分有趣的,尽管不难;我们以先根序沿着根为 α 的树移动,同时在 β 树中寻找匹配的名字,而且越过不可能出现对应元素的子树。(8)的名字 $A_0,\cdots,A_{n-1}$ 以相反的顺序 $A_{n-1},\cdots,A_0$ 被发现。

算法 C(寻找 CORRESPONDING 对)　给定 P0 和 Q0,它们分别指向 α 和 β 的数据表条目,这个算法逐次地找出满足上面提到的限制的(α',β')的指针对(P,Q)。

C1. [初始化] 置 P←P0,Q←Q0。(在本算法的剩余部分,指针变量 P 和 Q 将走遍分别有根 α 和 β 的树。)

C2. [是初等项吗?] 如果 CHILD(P) = Λ 或 CHILD(Q) = Λ,将(P,Q)作为所要求的对偶之一输出,并转到 C5。否则置 P←CHILD(P),Q←CHILD(Q)。(在这一步中,P 和 Q 指向满足(8)的项 α' 和 β',而且我们希望 MOVE α' TO β',当且仅当 α' 或 β'(或两者都)是初等项。)

C3. [匹配名字] (现在 P 和 Q 指向分别符合以下要求的数据项:

$$A_0 \text{ OF } A_1 \text{ OF } \cdots \text{ OF } A_{n-1} \text{ OF } \alpha$$

和

$$B_0 \text{ OF } A_1 \text{ OF } \cdots \text{ OF } A_{n-1} \text{ OF } \beta$$

目的是,通过考察组 A_1 OF ⋯ OF A_{n-1} OF β 中的所有名字,看能否使 $B_0 = A_0$。)如果 NAME(P) = NAME(Q),则转到 C2(匹配已被找到)。否则,如果 SIB(Q) ≠ Λ,置 Q←SIB(Q)并重复步骤 C3。(如果 SIB(Q) = Λ,则在这组中不出现匹配的名字,因此我们在步骤 C4 上继续。)

C4. [继续] 如果 SIB(P) ≠ Λ,则置 P←SIB(P)以及 Q←CHILD(PARENT(Q)),并转回到 C3。如果 SIB(P) = Λ,则置 P←PARENT(P)和 Q←PARENT(Q)。

C5. [完成了吗?] 如果 P = P0 则算法结束;否则转到 C4。

图 41 中示出这个算法的流程图。通过对于所涉及树的大小用归纳法,可以很容易地构造出这个算法正确的证明(见习题 9)。

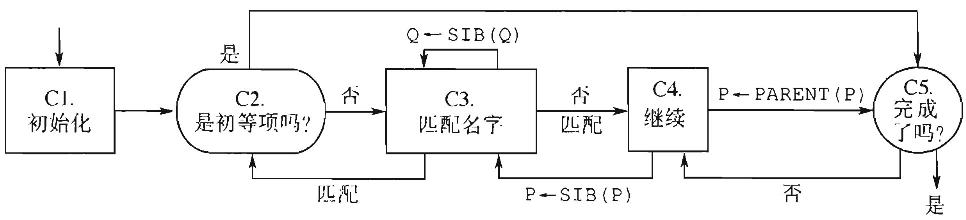


图 41　“MOVE CORRESPONDING”语句的算法

这时,值得研究算法 B 和 C 使用五个链接字段 PREV,PARENT,NAME,CHILD 以及 SIB 的方式。令人惊讶的特性是,算法 B 和 C 在沿数据表移动时,实际上做极小数量的工作,在这个意义下,这五个链接组成"完备的集合"。每当它们需要访问另一数据表条目时,它的地址立即就有了;根本不需要进行查找。如果在这个表中还有什么另外的链接信息,那将很难想像算法 B 和 C 如何可能做得再快些。(然而,见习题 11。)

每个链接字段可以看成是对程序的提示,插在其中以便使算法运行得更快些。(当然,构造诸表的算法,即算法 A,运行相应较慢,因为有更多的链接要填入。但构造表的工作只要做一次。)另一方面,显然,上面构造的数据表包含许多冗余的信息。让我们考虑如果删除某些链接字段,将发生什么情况。

尽管在算法 C 中不使用 PREV 链接,但它对于算法 B 却极其重要,而且似乎是任何 COBOL 编译程序必不可少的部分,除非要进行长时间的查找。因此一个把具有相同名称的所有项链接在一起的字段对于效率来说是必不可少的。我们或许可以稍微地修改策略并且采用循环链接以代替用 Λ 结束每个表,但是除非其它的链接字段被改变或者消去,否则绝无理由这样做。

在算法 B 和 C 中都使用了 PARENT 链接,尽管如果在算法 C 中使用辅助栈,或者增加 SIB,使得包括穿线链接(如同在 2.3.2 小节中那样),则可以避免使用它。所以我们看到,仅仅在算法 B 中,才以必不可少的方式使用 PARENT 链接。如果把 SIB 链接穿线使得现在有 SIB = Λ 的项将有 SIB = PARENT 代之,通过遵循 SIB 链接将有可能确定任何数据项的父亲的穿线位置;添加的穿线链接可以通过在每个节点中的新的 TAG 字段——它指出 SIB 链接是否穿线,或者通过这样的条件,即如果数据条目以其出现顺序连续地被保存在内存中,则"SIB(P) < P"——来加以区分。这将意味着在步骤 B5 中的简短查找是必要的,因此算法将相应地较慢些。

NAME 链接仅仅在步骤 B6 和 C3 中被这些算法所使用。在两种情况下,如果不出现 NAME 链接,我们可用其它方式做"NAME(S) = P_k"或"NAME(P) = NAME(Q)"的测试(见习题 10),但是这将大大地降低算法 B 和算法 C 的内循环的速度。这里我们再一次地看到链接的空间和算法的速度之间的折中。(当考虑 MOVE CORRESPONDING 的典型使用时,在 COBOL 编译程序中算法 C 的速度不是特别重要的;但算法 B 应当是快的。)经验指出,在一个 COBOL 编译程序内,也可以看到 NAME 链接的其它重要的用法,特别是在打印诊断信息时。

算法 A 逐步地构造数据表,而且它绝无把节点返回到可用存储池的情况;所以我们通常发现数据表条目以 COBOL 源程序中数据项出现的顺序来取连续的内存单元。因此在我们的例子(5)中,单元 A1,B3,…将彼此相跟随。数据表的这种顺序的本性导致了某种简化;例如,每个节点的 CHILD 链接或者是 Λ,或者它指向立即紧跟着的节点,所以可以把 CHILD 缩简成一位的字段。或者,如果 PARENT(P + c) = P,其中 c 是数据表中的节点大小,则 CHILD 可以被删除以有利于测试。

因此,尽管从算法 B 和 C 的速度的观点来看它们是有帮助的,但是五个链接字段并非全都必不可少。这种情况对于大多数多重链接结构来说是相当典型的。

有必要指出,在 20 世纪 60 年代初期,至少有一半写 COBOL 编译程序的人都独立地

走到使用五个链接(或者五个中的四个,通常是省去 CHILD 链接)保持数据表的同一路子上来。这种技术的第一个发表者是 H.W.Lawson,Jr.[*ACM National Conference Digest* (Syracuse,N.Y.:1962)]。但在 1965 年,David Dahm 给出了一种巧妙技术,只使用两个链接字段以及数据表的顺序存储,而不必很大地减低速度,就能实现算法 B 和 C 的效果。请见习题 12 至 14。

习 题

1.[*00*] 把 COBOL 数据配置当做树结构,由 COBOL 程序员列出的数据项是先根序,后根序,还是这两种次序都不是?

2.[*10*] 试对算法 A 的运行时间进行评述。

3.[*22*] PL/I 语言接受类似于 COBOL 的数据结构,只是层次号序列可以是任意的。例如,

1	A		1	A
3	B		2	B
5	C	等价于	3	C
4	D		3	D
2	E		2	E

一般地说,可把规则 a)修改如下:“一个组中的项必须有非递增的层次号码的序列,所有这些层次号码大于组名的层次号码。”对算法 A 应做什么修改将使它从 COBOL 的约定转化成为 PL/I 的这个约定?

▶4.[*26*] 算法 A 并不检测 COBOL 程序员是否违反正文中所述的规则 c)的错误。它应如何修改 A 使得只有满足规则 c)的数据结构才将被接受?

5.[*20*] 在实践中,可以对算法 B 提供符号表访问链接表作为输入,以代替我们所说的“P_0, P_1,…,P_n”。令 T 是使得

INFO(T) $\equiv P_0$,INFO(RLINK(T)) $\equiv P_1$,…,INFO(RLINK$^{[n]}$(T)) $\equiv P_n$,RLINK$^{[n+1]}$(T) = Λ

的指针变量,说明如何修改算法 B 使得它使用这样的链接表为输入。

6.[*23*] PL/I 语言接受和在 COBOL 中那些非常相似的数据结构,但对于规则 c)不作限制;代替的是,我们有以下规则,即如果合格的访问(3)显示“完备的”限定,则它是无二义性的——即是,如果对于 $0\le j<n$,A_{j+1} 是 A_j 的父亲,而且 A_n 没有父亲,则它是无二义性的。规则 c)现在变弱成为如下的简单条件,即组的两项没有相同的名字。(2)中的第 2 个“CC”,将无二义性地看成是“CC OF AA”,相对于刚才提出的 PL/I 的约定,三个数据项

```
1 A
    2 A
        3 A
```

将被看成“A”,“A OF A”,“A OF A OF A”。(注:实际上,在 PL/I 中,词“OF”被句点所代替,而且顺序是倒过来的;“CC OF AA”在 PL/I 中写成“AA.CC”,但对于现在的习题的目的说来这并不重要。)说明如何修改算法 B,使得它遵从 PL/I 的约定。

7.[*15*] 给定(1)中的数据结构,COBOL 语句“MOVE CORRESPONDING SALES TO PURCHAS-

ES"意味着什么?

8. [*10*] 按照正文中的定义,在什么情况下,"MOVE CORRESPUNDING α TO β"与"MOVE α TO β"完全相同?

9. [*M23*] 证明算法 C 正确。

10. [*23*] (a)如果在数据表节点中没有 NAME 链接,则如何实施在步骤 B6 中的测试"NAME(S) = P_k"? (b)如果在数据表条目中没有 NAME 链接,如何实施 C3 中的测试"NAME(P) = NAME(Q)?(假设所有其它链接像在正文中一样表示。)

▶ 11. [*23*] 在正文算法的策略中附加什么链接或改变可以使算法 B 或算法 C 运行得更快些?

12. [*25*] (D.M.Dahm) 考虑对于每个项仅使用两个链接

PREV　(如同正文中那样)

SCOPE　(与本组中最后的初等项的链接)

以顺序单元表示数据表。我们有 SCOPE(P) = P,当且仅当 NODE(P)表示初等项。例如,(5)的数据表将以下表代替:

	PREV	SCOPE		PREV	SCOPE		PREV	SCOPE
A1:	Λ	G4	F3:	Λ	G4	B5:	B3	B5
B3:	Λ	D7	G4:	Λ	G4	C5:	C7	G9
C7:	Λ	C7	H1:	Λ	G9	E9:	E3	E9
D7:	Λ	D7	F5:	F3	G8	D9:	D7	D9
E3:	Λ	E3	G8:	G4	G8	G9:	G8	G9

(试同 2.3.3 小节的(5)作比较。)注意 NODE(P)是 NODE(Q)之下树的一部分当且仅当 Q < P ≤ SCOPE(Q)。试设计一个算法,当数据表有这个格式时它实现算法 B 的功能。

▶ 13. [*24*] 当数据表有习题 12 中所示格式时给出替换算法 A 的算法。

▶ 14. [*28*] 当数据表有习题 12 中所示格式时给出替换算法 C 的算法。

15. [*25*] (David S.Wise) 重新阐述算法 A 使得栈不使用额外的存储。[提示:在现在的阐述中由栈所指所有节点的 SIB 字段是 Λ。]

2.5 动态存储分配

我们已经看到链接的使用如何意味着数据结构不必顺序地放置在内存中;许多表格可以独立地在公共的池状的内存区域中增长和收缩。然而,我们的讨论总是暗中假定所有节点都有相同的大小——即每个节点占有某个固定数目的内存单元。

对于大量的应用说来,可以找到一种合适的折中,使得对于所有结构确实使用一致的节点大小(例如见习题2)。习惯上,不是简单地取所必需的最大的节点大小因而在较小的节点处浪费空间,而是选择较小的节点大小并且应用所谓的经典的*链接存储原理*:"如果在这里没有存放信息的空间,那就把它放在别处,并安插一个指向该处的链接。"

然而,对于大量的其它应用,单一的节点大小是不合理的;我们经常希望大小变化的节点共享公共的内存区域;以另一个角度来谈这件事,我们要求算法从更大的存储区域保留和释放可变大小的内存块,其中这些块是由连续的内存单元组成的。这样的技术一般称为*动态存储分配*算法。

有时,通常在模拟程序中,我们需要对于较小大小的节点进行动态存储分配(比如从1到10个字);而在其它时候,通常是在操作系统中,我们主要地涉及较大的信息块。这两种视点导致对于动态存储分配的稍微不同的解决方法。为求得这两个方法之间的一致性,我们一般地在本节中使用*块*(*区*)和*区域*而不使用"节点",来表示一组连续的内存单元。

一些作者于1975年左右开始把可用的内存池称做"堆"。但在这套书中,我们仅仅在与优先队列有关的更传统的意义下使用该词(见5.2.3小节)。

A. 保留 图42示出典型的*内存图*或"跳棋盘",即示出某个内存池当前状态的图表。在这种情况下内存被划分成的53个存储块,其中有"保留"的,也有正在使用的,还有21块"自由的"或"可用的"未被使用。在动态存储分配已运行一段时间之后,计算机的内存看起来或许就像这个样子,我们的头一个问题是是来回答下面两个问题:

a)在计算机内部可用空间的这个分划是如何表示的?

b)给定可用空间的这种表示,找出 n 个连续自由空间的块并加以保留的好算法是什么?

当然,对于问题a)的回答,是在某个地方保留可用空间表;通过使用可用空间*本身*来包含这样的表几乎总可以做得最好。(例外的情况是当我们对于盘文件或其它存储器分配存储空间时,不一致的存取时间使得用分开的可用空间的目录表会更好些。)

因此,我们可以把可用的段*链接在一起*:每个自由存储区域的第一个字可以包含该块的大小和下一个自由区域的地址。自由块可以用大小的递增或递降的顺序链接在一起,或者以内存地址的顺序,或者以实际上随机的顺序来做。

例如,考虑图42,它示出从0到131071编址的131 072个字的内存。如果我们以内存单元的顺序把可用块链接在一起,我们将有指向第一个自由块的变量 AVAIL(在此情

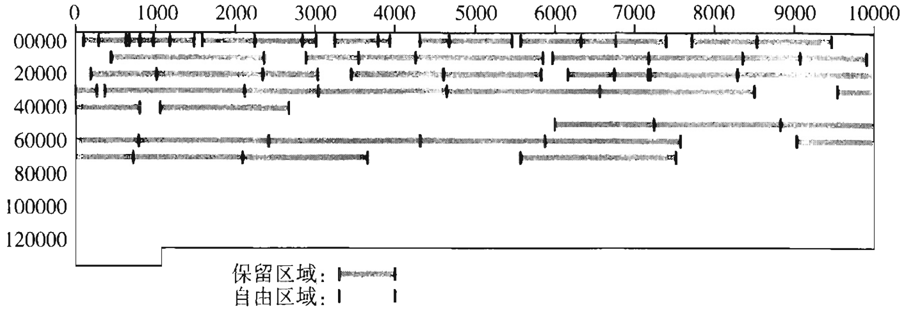


图 42　内存图

况下,AVAIL 将等于 0),其余的块将被表示如下:

位置	SIZE	LINK	
0	101	632	
632	42	1488	
⋮	⋮	⋮	[17 个相似的条目]
73654	1909	77519	
77519	53553	Λ	[对于最后链接的特殊标记]

因此单元 0 到 100 形成头一个可用块;在图 42 中示出的保留区域 101 ~ 290 和 291 ~ 631 之后,我们在单元 632 ~ 673 中有更多的自由空间;等等。

至于问题 b),如果我们需要 n 个连续的字,显然必须找出 $m \geq n$ 个可用字的某个块并把它的大小减小为 $m - n$。(而且,当 $m = n$ 时,我们还必须从这个表中删去该块。)可能会有好几个有 n 或更多个单元的块,因此问题就变成:应该选择哪个区域呢?

这个问题有两个主要答案:我们可以使用*最好适合方法*或*首先适合方法*。在前一种情况下,我们决定选择有 m 个单元的区域,其中 m 是所出现的等于 n 或更大的值中最小者。这可能要求在作出决策之前对整个可用空间表的查找。另一方面,首先适合方法,简单地选择有 $\geq n$ 个字的第一个遇到的区域。

从历史上说,最好适合方法被广泛使用好多年;这自然地看起来是好的决策,因为它节省更大的可用区域供给以后可能需要时使用。但是最好适合方法也有若干缺陷:它运行较缓慢,因为它涉及了相当长时间的查找;如果最好适合方法不是由于其它原因,比首先适合方法实质上更好,这额外的查找时间就不值得了。更重要的是,最好适合方法倾向于增加非常小的块的个数,而小块的扩散通常是不合意的。存在某些情况,其中首先适合技术确实比最好适合方法要更好些;例如,假设给了我们其大小为 1300 和 1200 的两个可用内存区域,并且假定有大小为 1000, 1100 和 250 的顺序内存请求:

内存请求	可用区域 首先适合方法	可用区域 最好适合方法	
——	1300,1200	1300,1200	(1)
1000	300,1200	1300,200	
1100	300,100	200,200	
250	50,100	卡住	

(相反的例子出现在习题 7 中。)关键之点是,没有哪一个方法明显地优越于另一个。因此简单的首先适合方法是可取的。

算法 A(*首先适合方法*) 令 AVAIL 指向第一个可用存储块,并且假设带有地址 P 每个可用块有两个字段:SIZE(P),即块中字的个数;LINK(P),即下一个可用块的指针。最后的指针是 Λ。这个算法查找和保留 N 个字的内存块,或者报告失败。

A1. [初始化] 置 Q←LOC(AVAIL)。(贯穿于这个算法,我们使用两个指针 Q 和 P,它们一般地通过条件 P = LINK(Q)而相互关联。我们假定 LINK(LOC(AVAIL)) = AVAIL。)

A2. [表结束了吗?] 置 P←LINK(Q),如果 P = Λ,则算法不成功地结束;没有可供 N 个连续字块的内存空间。

A3. [SIZE 足够吗?] 如果 SIZE(P)≥N,转到 A4;否则置 Q←P 并返回到步骤 A2。

A4. [保留 N] 置 K←SIZE(P) − N。如果 K = 0,则置 LINK(Q)←LINK(P)(由此从这个表删去空的区域);否则,置 SIZE(P)←K。这个算法成功地结束,并且保留由单元 P + K 开始长度为 N 的区域。 ▌

这个算法肯定是足够地直截了当的。然而,只通过策略上的微小的改变,就可作出在运行速度上的重大改进。这个改进是十分重要的,因此读者将会发现不告知这个秘密而自己去发现它是一件快事(见习题 6)。

无论对存储的分配是要求小的 N 还是大的 N,算法 A 都是可用的。然而,暂时让我们假定,我们主要是对于大的 N 值感兴趣。然后注意当在这个算法中 SIZE(P)等于 N + 1 时,发生什么情况。我们达到步骤 A4 并把 SIZE(P)减少成 1。换句话说,刚刚建立了大小为 1 的可用块;这个块是如此之小,因而实际上它是没有用的,它只不过阻塞着系统。如果我们保留 N + 1 字的整个块,而不是节省额外的字,我们可能处境更好。通常扩展内存的一些字以避免处理不重要的细节,是更好的。类似的说明也适用于当 K 很小时 N + K 个字的块。

如果我们允许保留多于 N 个字的可能性,则有必要记住已经保留了多少字,使得过后当这个块再次变成可用时,整个 N + K 个字的集合可予释放。这附加的簿记的数量意味着在每个块中我们都紧束空间以便仅当在某些情况下出现严紧的适合时,才使系统更加有效。所以这个策略似乎并不特别吸引人。然而,作为每个可变大小的块的第一个字的特殊控制字,由于若干原因通常证明是合乎要求的,因此期望,无论对于可用的还是保留的块,SIZE 字段出现在所有块的第一个字中,这通常并非不合理。

遵照这些约定,我们将修改上面的步骤 A4 如下:

A4′.［保留≥N］置 K←SIZE(P) - N。如果 K < c(其中 c 是小的正常数,选定它以反映在节省时间的利益下我们愿意牺牲的内存数量),置 LINK(Q)←LINK(P)和 L←P。否则置 SIZE(P)←K,L←P + K,SIZE(L)←N。算法成功地结束,并且保留了由单元 L 开始的长度为 N 或更大的区域。

尽管有很少的理论或经验的证据存在,以对 c 的选定值同其它值作比较,但是常数 c 的值用大约 8 或 10 是值得建议的。当使用最好适合方法时,K < c 的测试要比对首先适合更为重要,因为更严紧的适合(即较小的 K 值)更有可能出现,因此对此算法,可用块的个数应保持尽可能地小。

B. 释放　现在让我们考虑相反的问题:当内存块不再需要的时,我们应当怎样把它们返回到可用空间表中去?

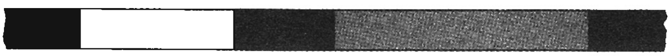
也许通过废料收集(参见 2.3.5 小节)会诱使我们草草打发这个问题;我们可以遵照以下决策行事,即在空间用完之前,什么也不做,而当空间用完后寻找当前正在使用的所有区域,并且制作出新的 AVAIL 表。

然而,废料收集的思想并非对所有应用都值得推荐。首先,如果我们能够保证当前正在使用的所有区域都将很容易找到,则我们就需要相当"守规矩"地使用指针。而这种规矩在这里所考虑的应用中通常是较缺少的;其次,如同我们以前所见到的,当内存已接近满了时废料收集趋于缓慢。

为什么废料收集的方法不是令人满意还有另一更重要的原因,它起因于前边关于这个技术的讨论中我们未曾碰到的一个现象:假设有两个相邻的内存区域,它们两个都是可用的,但由于废料收集原理,其中之一(以阴影示出)不在 AVAIL 表中。


(2)

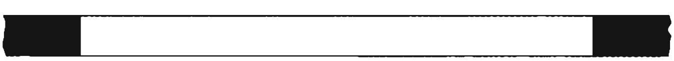
上图中,左端和右端的深阴影区域是不可用的。我们现在可以保留已知是可用区域的一部分:


(3)

如果这时废料收集出现,我们有两个分开的自由区域:


(4)

可用区域和保留区域之间的边界倾向于永久存在,而且随着时间的推移,情况逐渐变得更糟。但如果我们有只要块变为自由了就立即把它们返回到 AVAIL 表中这样一种原理,并且把相邻的可用区域都融合在一起,我们就把(2)融合成为


(5)

而且我们就会得到(6),这比(4)好得多。这个现象引起废料收集技术导致内存比应有的更为破碎。


(6)

为了消除这个困难,我们可以把废料收集和紧凑内存的过程一起使用,即把所有保留的内存块移动到相邻的单元,使得每当完成废料收集过程时,所有可用块就跑到一起来了。同算法 A 相对照,分配算法现在变成完全平凡了,因为在所有时刻下都只有一个可用块。即使这个技术要花时间重新复制当前正在使用的单元,并且改变其中链接字段的值,但当有对指针的守规矩的使用时,以及当在每块中有空闲的链接字段以供应废料收集算法使用时,它仍可以相当的效率被应用(见习题 33)。

由于许多应用不满足废料收集的可行性的要求,我们现在将研究把内存返回到可用空间表的方法。在这些方法中惟一的困难是融合的问题:即两个相邻的自由区域应当合并成一个。事实上,当由两个可用块限定的区域变为自由时,所有三个区域都应合并成一个。这样一来,*即使在长时间里,存储区域连续不断地被保留又被释放,仍然可以得到内存的好的平衡*。(关于这个事实的证明,请见以下的“百分之五十规则”。)

问题是确定在被返回块的任何一边的区域当前是否可用;如果它们是,我们就要适当地修改 AVAIL 表,这后一个操作比起设想的要更为困难。

对这些问题的第一种解决方法是以递增的内存地址来保持 AVAIL 表。

算法 B(通过排序表释放) 在算法 A 的假定下,并作附加的假定即 AVAIL 表是按内存单元排序了的(即,如果 P 指向一个可用块且 LINK(P) ≠ Λ,则 LINK(P) > P),这个算法往 AVAIL 表添加由单元 P0 开始的 N 个连续单元的块。我们自然假定这 N 个单元无一已是可用的。

B1. [初始化] 置 Q←LOC(AVAIL)。(见上面步骤 A1 中的说明。)

B2. [推进 P] 置 P←LINK(Q)。如果 P = Λ,或者如果 P > P0,则转到 B3;否则置 Q←P并重复步骤 B2。

B3. [校验上界] 如果 P0 + N = P 且 P ≠ Λ,则置 N←N + SIZE(P)并置 LINK(P0)←LINK(P),否则置 LINK(P0)←P。

B4. [校验下界] 如果 Q + SIZE(Q) = P0 (我们假定

SIZE(LOC(AVAIL)) = 0

因此这个检验在 Q = LOC(AVAIL)时总失败),置 SIZE(Q)←SIZE(Q) + N 及 LINK(Q)←LINK(P0)。否则置 LINK(Q)←P0,SIZE(P0)←N。 ▌

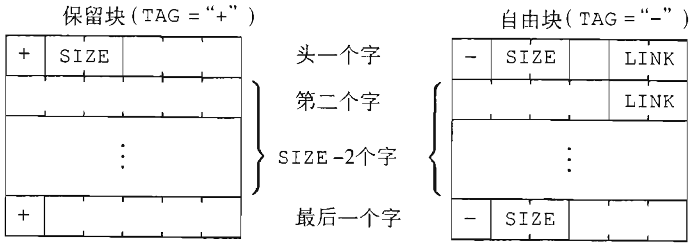
基于指针 Q < P0 < P 是三个连续可用区域的开始地址这样的事实,步骤 B3 和 B4 完成所要求的融合。

如果 AVAIL 表中不是以单元的顺序保持的,读者将看到融合问题的“硬算”方法需要对整个 AVAIL 表中的完全查找;算法 B 把这个通遍查找减少到平均大约是 AVAIL 表的一半(在步骤 B2)。习题 11 说明,如何修改算法 B 使得平均说来,只需查找 AVAIL 表的三分之一左右。但是显然,当 AVAIL 表很长时,所有这些方法都比我们想要的要慢得多。有没有其它的方法来保留和释放存储区域使得我们不必对整个 AVAIL 表进行

广泛的查找呢?

我们现在将考虑一种方法,当把存储返回时,消除所有的查找,而且像在习题 6 中那样,可加以修改以便当保留存储时,避免几乎所有的查找。这个技术利用在每个块两端的 TAG 字段和在每块第一个字中的 SIZE 字段;当使用相当大的块时,这个开销可以忽略不计,尽管当块有很小的平均大小时,这也许是要支付的相当大的代价。习题 19 中描述的另一种方法在每块的第一个字中只要求一个二进位,其代价仅是稍多一点的运行时间和稍微复杂一点的程序。

无论如何,让我们现在假定,我们不介意于稍微添加一点控制信息,以便当 AVAIL 表很长时,节省在算法 B 上的大量时间。我们将描述的方法假定每块有下列格式:



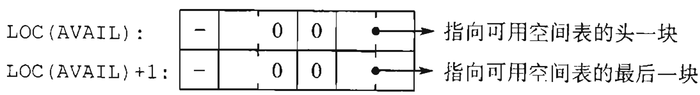
(7)

在下列算法中的思想是保持双重链接的 AVAIL 表,使得可以方便地从表的随机部分删去条目,在一个块的任一端的 TAG 字段可用于控制融合的过程,因为我们可以很容易地指出两个相邻的块是否可用。

通过令在第一个字中的 LINK 指向表中的下一个自由块,和令第二个字中的 LINK 指向前一个块,即可以熟知的方式实现双重链接。因此,如果 P 是可用块的地址,我们总有

$$\text{LINK}(\text{LINK}(\text{P})+1)=\text{P}=\text{LINK}(\text{LINK}(\text{P}+1)) \qquad (8)$$

为确保适当的"边界条件",把表头设置如下:



(9)

对于这个技术的首先适合保留算法可以被设计成同算法 A 非常相像,因此在这里我们不加考虑(请见习题 12)。这个方法主要的新特征,是内存块可以在实质上固定的时间内被释放的方式。

算法 C (利用边界标记释放)　假定单元的块有如(7)中所示的形式,并且假定 AVAIL 表如上面所述那样是双重链接。这个算法把以地址 P0 开始的单元的块放入 AVAIL 表中。如果可用存储池是从单元 m_0 开始直到 m_1(含)为止,为方便起见,本算法假定

$$\text{TAG}(m_0-1)=\text{TAG}(m_1+1)=\text{“}+\text{”}$$

C1.[检验下界]　如果 TAG(P0 − 1) = "+",转到 C3。

C2.[删去下部区域] 置 P←P0 − SIZE(P0 − 1),而后置 P1←LINK(P),P2←LINK(P + 1),LINK(P1 + 1)←P2,LINK(P2)←P1,SIZE(P)←SIZE(P) + SIZE(P0),P0←P。

C3.[检验上界] 置 P←P0 + SIZE(P0)。如果 TAG(P) = "+",则转到 C5。

C4.[删除上部区域] 置 P1←LINK(P),P2←LINK(P + 1),LINK(P1 + 1)←P2,LINK(P2)←P1,SIZE(P0)←SIZE(P0) + SIZE(P),P←P + SIZE(P)。

C5.[加到 AVAIL 表] 置 SIZE(P − 1)←SIZE(P0),LINK(P0)←AVAIL,LINK(P0 + 1)←LOC(AVAIL),LINK(AVAIL + 1)←P0,AVAIL←P0,TAG(P0)←TAG(P − 1)←"−"。 ▌

算法 C 的步骤是存储布局(7)的直截了当的结果;一种稍微快一点的也稍微长一点的算法出现于习题 15 中。如同在(9)中所示,在步骤 C5 中,AVAIL 是 LINK(LOC(AVAIL))的缩写。

C."伙伴系统" 我们现在将研究适合于二进制计算机使用的动态存储分配的另一种方法。这个方法在每块中使用一个二进位的开销,而且它要求所有块有 1,2,4,8 或 16 等的长度。如果一个块的长度不是对应于某个整数 k 的 2^k 个字的话,则选取 2 的下一个更高的次幂,而且相应地分配额外未使用的空间。

这个方法的思想,是保持每个大小为 2^k,$0 \le k \le m$ 的可用块的分开表。在分配之下整个内存空间也由 2^m 个字组成,我们可以假定它有从 0 开始到 $2^m - 1$ 的地址。原来,整个 2^m 个字都是可用的。过后,当要求 2^k 个字的一块时,如果没有这样大小的块可用,则把一个更大的可用块分开成两个相等的部分;最终,将出现一个正好是 2^k 大小的块。当把一个块分成两块(它们每一块都是原来块的一半那么大),这两块就称为伙伴(buddies)。过后当两个伙伴再次可用时,它们又合并回来成为一块;这样这个过程可以无限地维持下去,直到某个时刻我们用完内存为止。

奠定这个方法在实践上的有用性的关键事实是,如果我们知道一个块的地址(即它的头一个字的内存地址),而且如果我们还知道该块的大小,那我们就知道它的伙伴的地址。例如,开始于二进制单元 101110010110000 的大小为 16 的块的伙伴是开始于二进制单元 101110010100000 的一个块。为了知道为什么这为真,我们首先观察到,当算法进行时,*大小为2^k的一个块的地址是2^k的倍数*。换句话说,在二进制下的地址在右边至少有 k 个零。这个观察很容易通过归纳法来证明:如果它对于所有大小为 2^{k+1}的块为真,则当把逐个块折半时它肯定为真。

因此,一个大小为比如说 32 的块有形如 $xx\cdots x00000$ 的地址(其中 x 表示 0 或 1);如果把它分开,则新形成的伙伴系统有地址 $xx\cdots x00000$ 和 $xx\cdots x10000$。一般说来,令 $\mathrm{buddy}_k(x)$ = 大小为 2^k 其地址为 x 的块的伙伴的地址;我们求得

$$\mathrm{buddy}_k(x) = \begin{cases} x + 2^k, \text{如果 } x \bmod 2^{k+1} = 0 \\ x - 2^k, \text{如果 } x \bmod 2^{k+1} = 2^k \end{cases} \tag{10}$$

通过在二进制计算机上通常可找到的"异或"(有时称做"有选择的补"或"无进位加")指

令,可容易地计算这个函数;见习题 28。

伙伴系统利用在每个块中的一位的 TAG 字段:

$$\begin{aligned}&\text{TAG(P)} = 0, \quad \text{如果具有地址 P 的块被保留}\\&\text{TAG(P)} = 1, \quad \text{如果具有地址 P 的块是可用的}\end{aligned} \tag{11}$$

除了在所有块中存在的这个 TAG 字段(其不得被保留块的用户窜改)之外,可用块还有两个链接字段 LINKF 和 LINKB,它们是双重链接系统通常向前和向后的链接;而且当其大小为 2^k 时,它们还有一个 KVAL 字段来确定 k。以下算法利用表格单元AVAIL[0],AVAIL[1],…,AVAIL[m],它们分别作为大小为 $1,2,4,\cdots,2^m$ 的可用存储的表的表头。这些表是双重链接的,因此像通常一样表头包含两个指针(见 2.2.5 小节):

$$\begin{aligned}&\text{AVAILF}[k] = \text{LINKF(LOC(AVAIL}[k])) = \text{对 AVAIL}[k]\text{ 表尾部的链接}\\&\text{AVAILB}[k] = \text{LINKB(LOC(AVAIL}[k])) = \text{对 AVAIL}[k]\text{ 表首端的链接}\end{aligned} \tag{12}$$

开始时,在分配任何存储之前,我们有

$$\begin{aligned}&\text{AVAILF}[m] = \text{AVAILB}[m] = 0\\&\text{LINKF}(0) = \text{LINKB}(0) = \text{LOC(AVAIL}[m])\\&\text{TAG}(0) = 1, \quad \text{KVAL}(0) = m\end{aligned} \tag{13}$$

(表示开始于单元 0 处长度为 2^m 的单个可用块),且

$$\text{AVAILF}[k] = \text{AVAILB}[k] = \text{LOC(AVAIL}[k]),\text{对于 } 0 \leqslant k < m \tag{14}$$

(表示对于所有 $k < m$,长度为 2^k 的可用块的空表)。

从伙伴系统的这一描述,读者可以发现,在阅读以下给出的算法之前,自己亲自设计用于保留和释放存储区域的必要算法,是颇有兴味的。注意在这个保留算法中,诸块可相当容易地分成两半。

算法 R(伙伴系统的保留)　这个算法,利用以上所说明的伙伴系统之组织,找出并保留一个 2^k 个单元的块,或报告失败。

R1. [找块] 命 j 是在 $k \leqslant j \leqslant m$ 的范围内最小的整数,使得对于它 AVAILF[j] ≠ LOC(AVAIL[j]),即对它来说,大小为 2^j 的可用块的表不是空的。如果不存在这样的 j,则这个算法不成功地终止,因为没有足够大小的已知可用块来满足需要。

R2. [从表中撤消] 置 L←AVAILF[j],P←LINKF(L),AVAILF[j]←P,LINKB(P)←LOC(AVAIL[j])以及 TAG(L)←0。

R3. [需要分开?] 如果 $j = k$,则这算法终止(我们已经找到并保留了一个由地址 L 开始的可用块)。

R4. [分开] j 减 1。然后置 P←L + 2^j,TAG(P)←1,KVAL(P)←j,LINKF(P)←LINKB(P)←LOC(AVAIL[j]),AVAILF[j]←AVAILB[j]←P。(这就把一个大块分开并把不用的一半记入原来是空的 AVAIL[j]表中。)返回步骤R3。▍

算法 S（伙伴系统的释放） 利用以上阐明的伙伴系统的组织，这个算法把一个始于地址 L 的 2^k 个单元的块，恢复到自由存储中。

S1.［伙伴可用否？］置 $\mathrm{P} \leftarrow \mathrm{buddy}_k(\mathrm{L})$。（见等式(10)。）如果 $k = m$ 或如果 TAG(P) = 0，或者如果 TAG(P) = 1 且 KVAL(P) $\neq k$，则转到 S3。

S2.［同伙伴结合］置 LINKF(LINKB(P)) ← LINKF(P)，LINKB(LINKF(P)) ← LINKB(P)。（这就从 AVAIL[k]表撤消块 P。）然后置 $k \leftarrow k+1$，而且如果 P < L 则置 L←P。返回到 S1。

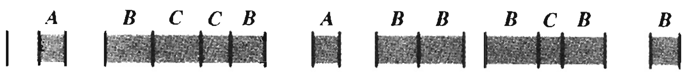
S3.［放到表中］置 TAG(L)←1，P←AVAILF[k]，LINKF(L)←P，LINKB(P)←L，KVAL(L)←k，LINKB(L)←LOC(AVAIL[k])，AVAILF[k]←L。（这就把块 L 放到 AVAIL[k]表中。） ▌

D. 诸方法之比较 要对这些动态存储分配算法进行数学分析，已经证明是十分困难的。但是，有一个有趣的现象，是相当容易来分析的，即“百分之五十规则”：

如果算法 A 和 B，以系统趋向于一个均衡状态这样一种方式，被连续地使用，其中在这个系统中平均有 N 个保留块，每一块都同样可能地成为下一个被释放者，而且其中算法 A 中之量 K 取非零值（或者，更一般地，如同在步骤 A4′中那样，取值 $\geq c$）之概率为 p，那么，可用块的平均数就近似地趋于 $\frac{1}{2}pN$。

这个规则告诉我们 AVAIL 表近似地将有多长。当数量 p 接近 1 时——如果 c 非常小而且如果块的大小不常彼此相等，则将发生这种情况——我们将有大约为不可用块之一半那么多的可用块；因此就有“百分之五十规则”之称。

不难来导出这个规则。考虑以下的内存图：



这就说明，保留块分成了三类：

A：当释放时，可用块的个数将减 1；

B：当释放时，可用块的个数将不变；

C：当释放时，可用块的个数将加 1。

现在命 N 是保留块的个数，并命 M 是可用块的个数；命 A，B 和 C 是上述标识类型的块数。我们有

$$N = A + B + C$$
$$M = \frac{1}{2}(2A + B + \epsilon) \tag{15}$$

其中 $\epsilon = 0, 1$ 或 2，取决于下界和上界的条件。

我们假定 N 实质上是常数，但 A，B，C 和 ϵ 是随机变量，在一个块被释放之后它们达到一个静止的分布，而在一个块被分配之后它们达到另一个（稍微不同的）静止的分

布。当一个块被释放时 M 的平均变化是 $(C-A)/N$ 的平均值;当一个块被分配时 M 的平均变化是 $1-p$。所以均衡假定告诉我们 $C-A-N+pN$ 的平均值为 0。但是 $2M$ 的平均值是 pN 加上 ϵ 的平均值,因为由(15),$2M=N+A-C+\epsilon$。从而得出百分之五十规则。

如果一个块的生存期是一个指数分布的随机变量,则关于每一个删除都适用于一个随机的保留块的假定将是正确的。另一方面,如果所有块有大约相同的生存期,则这个假定是假的;John E. Shore 曾经指出,当分配和释放倾向于有点先进先出的特性时,类型 A 的块显得比类型 C 的块"更老"些,因为相邻的保留块的一个序列倾向于是按从最年轻到最老的顺序的,而且还因为最新分配的块几乎总不是类型 A 的。这就倾向于产生更少量的可用块,并且给出比百分之五十规则的预测甚至更好的性能。[见 *CACM* **20** (1977), 812 ~ 820。]

关于百分之五十规则更详细的信息,请见 D. J. M. Davies, *BIT* **20** (1980), 279 ~ 288; C. M. Reeves, *Comp. J.* **26** (1983), 25 ~ 35; G. Ch. Pflug, *Comp. J.* **27** (1984), 328 ~ 333。

除了这个有趣的规则外,我们对这些动态存储分配算法的性能的知识,几乎完全以蒙特卡罗实验为基础。读者将会发现,当就一台特殊的机器和一个特殊的应用或一类应用,选择存储分配算法时,实施他们自己的模拟实验是有启发的。作者就曾在刚要写这一小节之前,进行了若干次这样的实验(其实,在作出这个百分之五十规则的证明之前,在做这些实验期间,就已经注意到这个规则)。这里,让我们简短地来检查一下这些实验的方法和结果。

基本的模拟程序运行如下,TIME 开始时为 0,而且内存区域开始时全都可用:

P1. TIME 增 1。

P2. 释放系统中在 TIME 的当前值时所有被调度为要释放的块。

P3. 利用第 3 章的一些方法,计算以某些概率分布为基础的两个量 S(一个随机的大小)和 T(一个随机的生存期)。

P4. 保留一个长度为 S 的新块,它是在(TIME + T)时要被释放的。返回到 P1。▌

每当 TIME 是 200 的一个倍数时,就打印出关于保留和释放算法之性能的详细统计。S 和 T 值的同一序列为每一对被测试的算法所使用。在 TIME 增加到超过 2000 之后,这个系统通常已差不多达到一种稳定状态,并明确指示此后将无限地保持此种状态。然而,依赖于可用存储的总量以及在步骤 P3 中 S 和 T 的分布,分配算法有时候将不能找出足够的空间,模拟实验将因此而被终止。

命 C 是可用内存单元的总数,并命 $\overline{S}$, $\overline{T}$ 表示步骤 P3 中 S 和 T 的平均值。容易看出,一旦 TME 足够大,在任何给定的时刻预期的内存不可用字数是 $\overline{ST}$。在实验中当 $\overline{ST}$ 大于大约 $\frac{2}{3}C$ 时,通常往往是在真正需要 C 个内存字之前就出现内存溢出。当块的大小相对于 C 来说很小时,则内存有可能充满到百分之九十以上,但是当允许块的大小超过 $\frac{1}{3}C$ 时(以至取更小的值时亦然),若事实上少于 $\frac{1}{2}C$ 个单元在使用,程序趋向于

认为内存已“填满”。经验的证据强烈地提示，如果预期进行有效的操作，则*大于$\frac{1}{10}C$的块大小不应为动态存储分配所使用*。

借助于百分之五十规则可以理解这一特性的原因：如果系统达到这样一个均衡条件，其中一个平均的自由块的大小 f 小于使用中的一个平均块的大小 r，则除非在紧急情况下有一个大的自由块可用，否则我们只能期望得到一个不能充满的请求。因此在一个不溢出的饱和系统中 $f \geqslant r$，而且我们有 $C = fM + rN \geqslant rM + rN \approx \left(\frac{1}{2}p+1\right)rN$。因此使用中的内存总量是 $rN \leqslant C\Big/\left(\frac{1}{2}p+1\right)$；当 $p \approx 1$ 时我们不可能使用多于大约$\frac{2}{3}$的内存单元。

实验以对于 S 的三种大小分布实施：

($S1$) 100 与 2000 之间均匀选取的整数。

($S2$) 分别以概率$\left(\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{16},\frac{1}{32},\frac{1}{32}\right)$选定的大小(1,2,4,8,16,32)。

($S3$) 以相等的概率选择的大小(10,12,14,16,18,20,30,40,50,60,70,80,90,100,150,200,250,500,1000,2000,3000,4000)。

时间分布 T 通常是对于固定的 t = 10，100 或 1000，在 1 与 t 之间均匀地选取的随机整数。

实验也可以这样来进行，即在步骤 P3 中，在 1 与$\min\left(\left\lfloor \frac{5}{4}U \right\rfloor, 12500\right)$之间均匀地选择 T，其中 U 是直到系统中的某个当前被保留的块下一次被调度释放为止，剩下的时间单位数。这种时间分布是用来模拟一个“几乎是后进先出的”行为：因为如果总是把 T 选择成 $\leqslant U$，则存储分配系统就将退化成为只不过是一个不需要复杂算法的栈操作(见习题 1)。所述的分布使 T 有大约百分之二十的时间被选择成大于 U，所以我们就几乎有(而不是完全有)一个栈操作。当采用这种分布时，诸如 A，B 和 C 这些算法，要比通常表现得好得多。在整个 AVAIL 表中很少有多于两个的项目，而同时却有大约 14 块保留块。另一方面，当使用这种分布时，伙伴系统算法 R 和 S 是较慢的，因为它需要更经常地以一种类似栈的操作来分开和合并诸块。这种时间分布的理论性质，显得特别难推导(见习题 32)。

这一节开头的图 42，是在 TIME = 5000 时的内存配置，用的是大小分布($S1$)和在{1,…,100}中一致分布的时间，并且利用像上述算法 A 和 B 中那样的首先适合的方法。对于这一实验，进入“百分之五十规则”之概率 p 实质上是 1，所以我们可以预期大约有保留块的一半那么多的可用块。实际上，图 42 说明有 21 块可用块和 53 块保留块。这并不否定百分之五十规则：例如，在 TIME = 4600 时有 25 个可用块和 49 个保留块。图 42 中的配置只不过说明了百分之五十规则怎样以统计的变化为条件。可用块的块数范围一般是在 20 与 30 之间，而保留块的块数一般是在 45 与 55 之间。

图 43 示出了使用与图 42 相同的数据，但使用的是“最好适合”而不是“首先适合”方法，所得到的内存图。步骤 A4′中的常数 c 选择成 16，以消除小块，而且结果概率 p 降低到约 0.7，并有更少的可用区域。

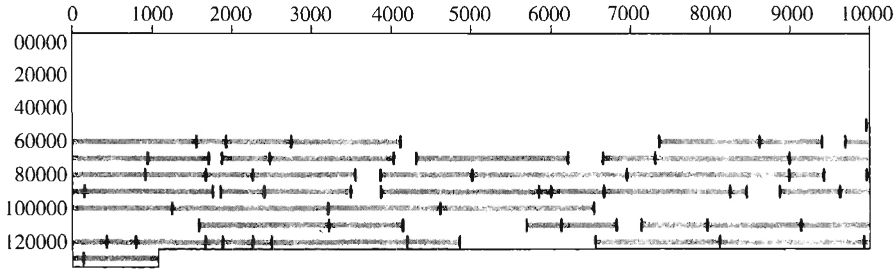


图 43　用“最好适合”方法得到的内存图(与示出“首先适合”方法的图 42,以及对于同一个存储请求序列示出“伙伴系统”的图 44 相比较)

当时间分布是从 1 到 1000 变化而不是从 1 到 100 时,就得到确切地类似于图 42 和 43 中所示的情况,而且所有相应的数量都近似地乘以 10。例如,有 515 个保留块;而在图 42 的等价条件下有 240 个自由块,在图 43 的等价条件下有 176 个自由块。

在最好适合与首先适合方法的所有实验中,后者总是显得更好些。当内存大小被穷尽时,在大多数情况下,在内存出现溢出之前,首先适合方法实际上比最好适合方法更能长久地持续活动。

伙伴系统也可应用到导致图 42 和图 43 的相同的数据上,从而得到图 44 作为其结果。这里,所有在 257 到 512 范围内的大小都被处理成 512,而所有在 513 与 1024 之间的大小都被提高到 1024,等等。平均说来,这意味着大约要求三分之四那么多的内存(见习题 21);当然,伙伴系统对于像上边(*S2*)那样的大小分布,而不是(*S1*)那样的,要工作得更好。注意在图 44 中有大小为 $2^9,2^{10},2^{11},2^{12},2^{13}$和 2^{14}的可用块。

对伙伴系统的模拟说明,它的执行比预料的要好得多。显然,伙伴系统有时将允许两个相邻的区域都是可用的,而不把它们合并成一个(设想它们不是“伙伴”);但这种情况在图 44 中不出现,事实上,这在实践中是少见的。在出现内存溢出的情况下,内存百分之九十五被保留,而这反映了一种十分好的分配平衡。况且,很少有必要在算法 R 中分开诸块,或者在算法 S 中合并它们;对于在最普遍使用的诸层上的可用块,剩下的树极像图 44。有助于理解在树之最低层上这种特性的某些数学结果,已经由 P. W. Purdom, Jr 和 S. M. Stigler 得到,见 *JACM* **17** (1970),683 ~ 697。

还有另一件令人吃惊的事情,就是算法 A 在如习题 6 所述修改之后,所得到的出色特性;平均说来,仅需要对可用块大小做 2.8 次检查(使用大小分布(*S1*)和在 1 与 1000 之间均匀选取的时间),而且多一半的时间,仅仅需要极小的值,即一次迭代。尽管事实上大约有 250 个可用块,但这却是真的。对于未修改的算法 A,相同的实验证明,平均大约需要 125 次迭代(所以每次大约要检查 AVAIL 表的一半);而且发现 20% 的时间需要 200 次或更多次的迭代。

事实上,未修改的算法 A 的这种行为,可以作为百分之五十规则的一个推论而预测到。在均衡状态下,内存中包含最后一半保留块的内存部分,也将包含最后一半的自由块;当一块被释放时,该部分将有一半时间被卷入,从而它必须被卷入到一半的分配

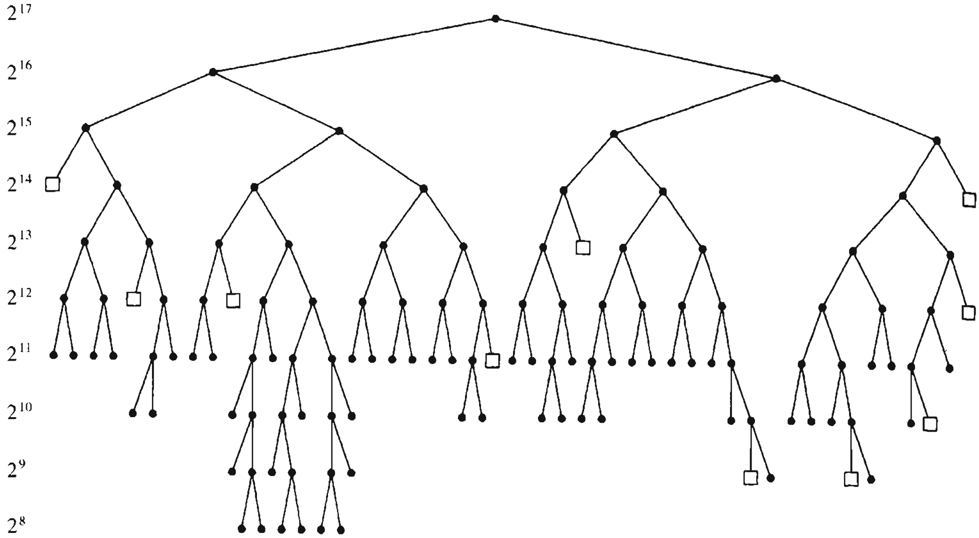


图 44 用“伙伴系统”得到的内存图(这个树结构指出了某些大块分成大小折半的“伙伴”的划分。正方形表示可用块)

中以便维持均衡。当以任何其它的分数来代替一半时,同样的论证亦成立。(这些发现属于 J. M. Robson。)

后面的习题包括两个主要方法的 MIX 程序,推荐它们作为上述说明的推论:(i)习题 12 和习题 16 中所修改的边界标记系统;以及(ii)伙伴系统。下面是近似结果:

	用于保留的时间	用于释放的时间
边界标记系统	$37+7A$	18,29,31 或 34
伙伴系统	$19+25R$	$27+26S$

这里,$A \geqslant 1$ 是在检索一个足够大的可用块时所需要的迭代次数;$R \geqslant 0$ 是一块分成二块的次数(在算法 R 中的 $j-k$ 之初始差);而 $S \geqslant 0$ 是在算法 S 期间伙伴块被重新融合在一起的次数。模拟实验表明,在所述的假定下,用大小分布(*S1*)以及在 1 与 1000 之间选定的时间,平均说来,我们可以取 $A=2.8, R=S=0.04$。(当代换成如前所述的“几乎是后进先出”的时间分布时,则发现这个平均值为 $A=1.3, R=S=0.9$。)这说明,两个方法都十分快,而且伙伴系统在 MIX 的情况下稍微更快些。记住,当块的大小不被限制为 2 的幂次时,伙伴系统要求多出大约 44%的空间。

对于习题 33 的废料收集和紧凑算法,其相应的时间估计是,大约要 104 个时间单位来找出一个自由节点,假定当内存近于半数充满时出现废料收集,并且假定诸节点有 5 个字的平均长度,又每个节点有 2 个链接。废料收集的正反两个方面,都已在 2.3.5 小节中讨论过了。当内存未被大量装入而且当满足适当的限制时,废料收集和紧凑算法是非常有效的;例如,在 MIX 计算机上如果内存空间绝不会充满到 1/3 以上,而且节点又相当小,废料收集方法要比其它两个方法都要快。

如果奠定废料收集的基础的假定被满足，最好的策略可能是把内存池分成两半并且在一半之内进行所有的分配。我们不在块变成可用时释放它们，而是简单地等候到内存当前活动的一半变满为止；然后我们可以把所有活动的数据都复制到另一半去，同时删去块间的所有空洞，使用类似于习题 33 给出的方法。当我们从一半转到另一半时，我们也可对每一半池的大小作调整。

以上提到的模拟技术也可应用于某些其它的存储分配算法。与这一小节的算法比较起来，其它的算法就显然太拙劣了，因此在这里仅仅简单地提及：

a)对每个大小保存独立的 AVAIL 表。一个单一的自由块必要时偶尔也分成两个更小的块，但并不试图再把这样的块重又合到一起。内存图破碎为越来越细的部分，直至濒于细如毫毛的惨状；一个与此相像的简单的方案，几乎相当于在分散的区域里做独立的分配，对于每个块的大小都分配一个区域。

b)企图进行"二级"分配：把内存分成 32 个大段。用"蛮干"分配方法来保留 1，2 或 3 份(很难再多了)相邻段的大块。每个这样的大块又被进一步分开以满足存储要求，直到在当前的大块内已不再剩有余地为止；而后保留另一大块供其后的分配使用。仅当每一大块内的所有空间都已成为可用时，才把它恢复到自由存储。这个方法几乎总是非常快地用完存储空间。

尽管这一特殊的"二级"分配方法，对于作者的模拟实验中所考虑的数据是失败的，却存在有其它一些情况(在实践中不经常出现)，其中多级分配技巧可能有益。例如，如果一个相当大的程序在若干阶段中运行，我们可能知道仅仅在某个子程序内需要某种类型的节点。一些程序也可能发现对于不同类的节点使用非常不同的分配策略，也是所希望的。通过带区来分配存储，在每个带区采用可能是十分不同的策略以及能一次释放整个带区，这样的思想是由 Douglas T. Ross 在*CACM* **10** (1967)，481 ~ 492 中予以讨论的。

关于动态存储分配进一步的实验结果，参见如下论文：*CACM* **12** (1969)，365 ~ 369，372；P.W.Purdom，S.M.Stigler 和 T. O. Cheam，*BIT* **11** (1971)，187 ~ 195；B. H. Margolin，R. P. Parmelee 和 M. Schatzoff，*IBM Systems J.* **10** (1971)，283 ~ 304；J. A. Campbell，*Comp. J.* **14** (1971)，7 ~ 9；John E. Shore，*CACM* **18** (1975)，433 ~ 440；Norman R. Nielsen，*CACM* **20** (1977)，864 ~ 873。

*** **E. 分布适合**　如果事先已经知道诸块大小的分布，而且不管块分配何时进行，当前的每个块都同样可能成为下一个被释放的块，则通过遵循 E. G. Coffman，Jr. 和 F. T. Leighton[*J. Computer and System Sci.* **38** (1989)，2 ~ 35]的建议，我们可以使用比迄今描述的通用技术要好得多的一项技术。他们的"分布适合方法"通过把内存分成大约 $N+\sqrt{N}\lg N$ 个槽来工作，其中 N 是在稳定状态下要加以处理的合适的极大块个数。每个槽有一个固定的大小，尽管不同的槽可以有不同的大小；其要点是，任何给定的槽，都有固定的边界，而且它将或者是空的或者包含单个已分配的块。

在 Coffman 和 Leighton 的方案中，头 N 个槽是按照假定的大小分布来放置的，而后边的$\sqrt{N}\lg N$ 个槽全都有极大的大小。例如，如果我们假定块的大小将均匀地分布在 1

和 256 之间，而且如果我们预期处理 $N=2^{14}$ 个这样的块，则我们将把内存分成大小为 1，2，…，256 的 $N/256=2^6$ 个槽，后边接着是包含 $\sqrt{N}\lg N=2^7\cdot 14=1\,792$ 个大小为 256 的块。当这个系统以其完全的容量进行操作时，我们预期它处理平均大小为 257/2 的 N 个块，并占有 $\frac{257}{2}N=2^{21}+2^{13}=2\,105\,344$ 个单元；这是我们分配给头 N 个槽的空间数量。我们还设置了另外的 $1792\cdot 256=458\,752$ 个单元来处理随机变化的效果；这额外的开销总计为整个空间的 $O(N^{-1/2}\log N)$，而不是像在伙伴系统中那样是 N 的一个常数倍，因此当 $N\to\infty$ 时它变成可忽略的因子。然而，在我们的例子中，它仍然占有总的分配的大约 18%。

应该把槽按顺序安排，使得较小的槽在较大的槽前边。给定这种安排，我们就可以通过使用首先适合或最好适合技术来分配诸块。（在这种情况下，两种方法等价，因为槽的大小是有序的。）在我们的假定之下，效果是每当一个新的分配请求到来时，就在头 N 个槽当中开始在实质上随机的位置进行查找，而且继续进行直到我们找到一个空的槽为止。

如果对于每个查找，开始的槽真正是在 1 和 N 之间随机的，我们就不会很经常地要侵犯溢出区域。其实，如果在随机槽处恰好插入 N 个项，则平均说来溢出将仅出现 $O(\sqrt{N})$ 次。原因是我们可以把这个算法和带线性探查（算法 6.4L）的散列法作比较。线性探查除了对于一个空单元的查找将从 N 绕回到 1 而不是进入到溢出区域之外，有着相同的特性。在定理 6.4K 中对于算法 6.4L 的分析表明，当插入 N 个项时，每个项同它的散列地址的平均偏离是 $\frac{1}{2}(Q(N)-1)\sim\sqrt{\pi N/8}$；由循环对称性容易看出这个平均与对于每个 k，从槽 k 到 $k+1$ 的一个查找的平均次数相同。在分布适合方法中的溢出对应于从槽 N 到槽 1 的查找，只是我们的情况甚至更好，因为通过不绕转回来可避免某些拥塞现象。因此平均说来，将出现少于 $\sqrt{\pi N/8}$ 的溢出。这个分析不把删除考虑在内，仅当删除介于开始槽和已分配槽之间的另一个块时（见算法 6.4R），我们把块移回的话，删除才保持算法 6.4L 的假定。然而再把它们移回去，将只会增加溢出的机会。我们的分析也不能考虑一次有多于 N 个块存在的效果；如果我们仅假定块之间的抵达时间大约是驻留时间的 N 分之一的话，这个情况可能发生。对于多于 N 个块的情况，我们需要扩充对于算法 6.4L 的分析，但是 Coffman 和 Leighton 证明，溢出区域将几乎不会超过 $\sqrt{N}\lg N$ 个槽；对于所有的 M，跑出末尾的概率少于 $O(N^{-M})$。

在我们的例子中，在一个分配期间查找的开始槽不是在槽 1，2，…，N 当中均匀的，而是在槽 1，65，129，…，$N-63$ 当中均匀的，因为对于每一个大小，有 $N/256=64$ 个槽。但是这种对上一段中考虑的随机方法的偏离，使溢出甚至比预期的更为不可能。当然，如果关于块大小的分布和占有时间的假定被违反的话，所有的打赌都要输了。

F. 溢出 当不再有空间可用时，我们应该做些什么呢？假设，有一个请求，比如说，要求 n 个连续的字，而此时所有可用的块都太小。这种情况头一次发生时，通常有多于 n 个的可用单元存在，但它们不是连续的；“紧凑内存”（即移动某些使用中的单

元,使得所有可用单元都被挪到一起)将意味着我们可以继续处理。但紧凑是缓慢的;而且绝大多数情况是,当"首先适合"方法穷尽所有余地时,不论进行多少次紧凑和再紧凑,实际上此后不久空间也就完全用光了。因此,除非是在与废料收集相联系的特殊情况下,如同习题 33 中那样,一般说来已不值得来写紧凑程序。如果预期会出现溢出,则可以使用从内存撤消某些项目并把它们转存到一个外部存储装置上的某种方法,同时提供必要时再次把信息送回的条件。这就意味着,所有访问动态存储区域的程序,就其对其它块所作的访问来说,必须加以严格的限制,而且一般说来,要求特殊的计算机硬件(例如,缺乏数据时的中断,或自动"分页")在这些条件下能进行有效的操作。

为了判定哪些块是最可能撤消者,就必须有某些判定过程。一种想法是维持一份保留块的双重链接表,每当其中某个块被访问时,就把它移动到该表的前头;然后,诸块即以它们最近访问的次序,进行了有效的排序,而且在这个表的末尾的块是首先要撤消的块。通过把保留块放进一个循环表中,并且在每一块中包括一位"最近使用"位,就可以更简单地达到一种类似的效果;每当一块被访问时,该位即置成 1。当要撤消一个块的时候,一个指针沿着这个循环表移动,复置所有的"最近使用"位成 0,直到找到了自从指针上一次到达圆周上这个部分以来未曾使用过的一块为止。

J. M. Robson 已经证明[*JACM* **18** (1971),416 ~ 423],从不对保留块重新定址的动态存储分配策略,不可能保证有效地使用内存;总是会有使这种方法崩溃的病理情况。例如,甚至当诸块大小均被限制为 1 和 2 时,不论使用什么分配算法,内存仅仅约 2/3 被填满,就可出现溢出!习题 36 ~ 40 及 42 ~ 43 综述了 Robson 的有趣结果,在习题 42 ~ 43 中,Robson 证明,通过和首先适合方法作比较,最好适合方法有一个非常坏的最糟情况。

G. 继续阅读　基于比作者编写上述材料来有更多经验材料可资利用,Paul R. Wilson,Mark. S. Johnstone,Michael Neely 和 David Boles 编写了关于动态存储分配技术的全面综述和中肯评论。

习　题

1.[*20*] 如果存储请求总是以"后进先出"的方式出现,即只有所有被陆续保留的块都已经被释放以后,才有自由的保留块,问对于这一节的保留和释放算法,可以作些什么简化?

2.[*HM23*](E. Wolman) 假设我们要为可变长度的项目选择固定的节点大小,且又假设当每个节点有长度 k 而一个项目有长度 l 时,就采用$\lceil l/(k-b)\rceil$个节点来存这个项目。(这里 b 是一个常数,表示每个节点的 b 个字包含控制信息,比如指向下一节点的链接。)如果一项的长度 l 的平均值是 L,则 k 应如何选择,以使所要求的存储空间的平均数量为极小?(假定对于任何固定的 k,当 l 变化时,$(l/(k-b)) \bmod 1$ 的平均值等于$\frac{1}{2}$。)

3.[*40*] 通过计算机模拟,试比较存储分配的最好适合、首先适合与最坏适合方法;在后一种方法下,总是选择最大的可用块。在内存的用法方面,有什么显著的差别?

4.[*22*] 为算法 A 编写一个 MIX 程序,特别要着眼于力求内循环运行快。假定 SIZE 字段是(4:5),LINK 字段是(0:2),以及 $\Lambda < 0$。

▶ 5.[*18*] 假设已知 N 在算法 A 中总是 100 或更大。则在修改的步骤 A4′中置 $c = 100$ 是个好

主意吗？

▶ 6.[*23*]（下一个适合）在算法 A 已被反复地使用之后，有一种强烈的趋向，把 SIZE 小的块保存在 AVAIL 表的前头，以致通常在找到一个长度为 N 或更大的块之前，必须远远地进入这张表来进行检索。例如，注意在图 42 中，对于保留块和自由块两者，从内存的开始到末尾，诸块的大小实质上是如何在增加的。（在准备图 42 的同时，所用的 AVAIL 表已保持了按单元顺序的排序，如同算法 B 所要求的那样。）你能不能提出一个修改算法 A 的方法，使得(a)短小的块不趋向于累积于一个特殊的区域，以及(b)AVAIL 表仍然可以以内存单元递增的顺序来保持，以利于诸如算法 B 那样的一些算法？

7.[*10*] 例(1)说明有时候“首先适合”方法确实是优越于“最好适合”方法。试给出一个类似的例子，以示出一种“最好适合”方法优于“首先适合”方法的情况。

8.[*21*] 说明怎样以一种简单的方式修改算法 A，以得到关于“最好适合”方法的算法，来代替“首先适合”方法。

▶ 9.[*26*] 用“最好适合”方法，以什么方式来设计一个保留算法，使其无需检索整个 AVAIL 表？（试尽可能多地想出能减少必要检索的方法。）

10.[*22*] 说明怎样修改算法 B，使得从单元 P0 开始的 N 个连续单元的块成为可用的，无须假定这 N 个单元中的每一个当前不可用；事实上，假定这个正被释放的区域实际上可以与已经被释放的若干块相重叠。

11.[*M25*] 证明，在习题 6 的答案中所提议的对算 A 的改进，也可用来实现对算法 B 的一点改进，它把平均检索长度由 AVAIL 表长度的一半减少到这个长度的 1/3。（假定这个被释放的块将被插入到已分类的 AVAIL 表内的一个随机的位置处。）

▶ 12.[*20*] 试修改算法 A，使得它遵从(7)～(9)的约定，使用正文中所述的修改的步骤 A4′，并且也加入习题 6 中所作的改进。

13.[*21*] 为习题 12 的算法编写一个 MIX 程序。

14.[*21*](a)如果 SIZE 字段不出现在一自由块最后的字中，或者(b)如果 SIZE 字段不出现在一保留块的头一个字中，则对于算法 C 和习题 12 的算法将产生什么影响？

▶ 15.[*24*] 以稍微长些的程序为代价，在 TAG(P0 − 1)和 TAG(P0 + SIZE(P0))是正还是负的四种情况的每一种情况下，除绝对必要外不改变任何更多的链接，试说明怎样提高算法 C 的速度。

16.[*24*] 利用习题 15 的思想为算法 C 编写一个 MIX 程序。

17.[*10*] 当不出现可用块时，(9)中的 LOC(AVAIL)和 LOC(AVAIL) + 1 的内容将是什么？

▶ 18.[*20*] 图 42 和图 43 是利用相同的数据，以及实际上相同的算法（算法 A 和 B）而得到的，只是图 43 是通过把算法 A 加以修改以选择“最好适合”以代替“首先适合”而作出的。为什么这就使得图 42 有一块大的可用区域在内存的较高单元中，而在图 43 中则有一块大的可用区域在较低的单元中呢？

▶ 19.[*24*] 假设内存中诸块都有(7)的形式，但在块的最后一个字中没有所需要的 TAG 或 SIZE 字段。进一步假设，下述简单的算法正被用来再次地释放一个保留块：Q←AVAIL，LINK(P0)←Q，LINK(P0 + 1)←LOC(AVAIL)，LINK(Q + 1)←P0，AVAIL←P0，TAG(P0)←“ − ”。（该算法在把相邻的区域融合成为一体方面无所作为。）

设计一个类似于算法 A 的保留算法，它在检索 AVAIL 表的同时对相邻的自由块进行必要的融合，同时避免对内存进行(2)，(3)和(4)中那样不必要的分段。

20.[*00*] 为什么希望在伙伴系统中的 AVAIL[k]表都是双重链接的,而不是简单的直接的线性表?

21.[*HM25*] 考察当 n 趋于无穷时的比率 a_n/b_n,其中 a_n 是 $1+2+4+4+8+8+8+8+16+16+\cdots$ 的头 n 项之和,而 b_n 是 $1+2+3+4+5+6+7+8+9+10+\cdots$ 的头 n 项之和。

▶ 22.[*21*] 正文反复地说到,伙伴系统仅只允许使用大小为 2^k 的块,而且习题 21 表明这可能导致所需存储实质性地增加。但如果在同伙伴系统相关联中需要一个 11 个字的块,为什么我们不能找一个 16 个字的块,并把它分成为一个 11 个字的和两个分别为 4 个字和 1 个字的自由块呢?

23.[*05*] 二进地址为 011011110000,大小为 4 的块,其伙伴的二进地址是什么? 当块的大小为 16 而不是 4 又将如何?

24.[*20*] 按照正文中的算法,(大小为 2^m 的)最大的块没有伙伴,因为它表示存储的全部。如果定义 $\mathrm{buddy}_m(0)=0$(即命这一块就是它自己的伙伴),从而避免在步骤 S1 中对 $k=m$ 的测试,这对吗?

▶ 25.[*22*] 鉴定下述想法:"在实际情况下,使用伙伴系统的动态存储分配将从不保留大小为 2^m 的一个块(因为这将填满整个内存),而且一般说来,有一个极大值 2^n,没有比其更大的块被保留。因此,当结合后的块有大于 2^n 的大小时,在算法 S 中以这样大的可用块开始,还要把伙伴结合起来,将是浪费时间的。"

▶ 26.[*21*] 试阐明伙伴系统如何可用于从内存单元 0 到 M - 1 之间的动态存储分配,即便 M 不像正文中所要求的那样有 2^m 之形式。

27.[*24*] 为算法 R 写出一个 MIX 程序,并确定它的运行时间。

28.[*25*] 假定 MIX 是一台二进计算机,并有如下定义的一个新操作码 XOR(用 1.3.1 小节的记号):"C = 5,F = 5。对应于单元 M 中每个等于 1 的位,在寄存器 A 的对应位上取补(把 0 变成 1 或 1 变成 0);rA 的符号不受影响。执行时间为 $2u$。"

试为算法 S 写出一个 MIX 程序,并确定它的运行时间。

29.[*20*] 如果在每个保留块中没有标记位,伙伴系统能工作吗?

30.[*M48*] 若给定关于存储请求序列的合理分布,试分析算法 R 和 S 的平均特性。

31.[*M40*] 利用斐波那契序列代替 2 的幂,能否设计一个类似于伙伴系统的存储分配系统?(这样,我们可以从 F_m 个可用字开始,而且可以把有 F_k 个字的一个可用块分开成为长度分别为 F_{k-1}和 F_{k-2}的两个伙伴。)

32.[*HM46*] 试确定 $\lim_{n\to\infty}\alpha_n$,如果它存在的话,其中 α_n 是在如下定义的一个随机序列中 t_n 的平均值:给定对于 $0\leqslant k<n$ 的 t_k 的值,命 t_n 是从$\{1,2,\cdots,g_n\}$中均匀选取的,其中

$$g_n=\left\lfloor \frac{5}{4}\min(1000,f(t_{n-1}-1),f(t_{n-2}-2),\cdots,f(t_0-n))\right\rfloor$$

而且如果 $x>0$,则有 $f(x)=x$,如果 $x\leqslant 0$,则 $f(x)=\infty$。(注:某些有限的经验测试表明,α_n 可能近似于 14,但这大概不是很精确的。)

▶ 33.[*28*] (*废料收集和紧凑*) 假定内存单元 1,2,…,AVAIL - 1 正被用作有如下形式的可变大小的节点的一个存储池:NODE(P)的头一个字含有字段

SIZE(P) = NODE(P)中的字数;

T(P) = NODE(P)中链接字段的个数;T(P) < SIZE(P);

LINK(P) = 仅在废料收集期间使用的特殊的链接字段。

在内存中紧接 NODE(P)之后的是节点 NODE(P + SIZE(P))。假定,NODE(P)中仅有的用来链到其它节点的链接字段是 LINK(P + 1),LINK(P + 2),…,LINK(P + T(P)),而且这些链接字段中的每一个,或者是 Λ,或者是另一个节点的头一个字的地址。最后,假定在程序中另有一个进一步的链接变量,称做 USE,它指向这些节点之一。

试设计一个算法,它能:(i)确定出可由变量 USE 直接地或间接地访问的所有节点,(ii)改变所有链接使得保持结构关系,对于某个 K,把这些节点移入内存单元 1 到 K - 1;以及(iii)置 AVAIL←K。

例如,考虑内存的下列内容,其中 INFO(L)表示单元 L 除 LINK(L)外的内容:

1:SIZE = 2,T = 1	6:SIZE = 2,T = 0	AVAIL = 11
2:LINK = 6,INFO = A	7:CONTENTS = D	USE = 3
3:SIZE = 3,T = 1	8:SIZE = 3,T = 2	
4:LINK = 8,INFO = B	9:LINK = 8,INFO = E	
5:CONTENTS = C	10:LINK = 3,INFO = F	

你的算法应当把它变换成

1:SIZE = 3,T = 1	4:SIZE = 3,T = 2	AVAIL = 7
2:LINK = 4,INFO = B	5:LINK = 4,INFO = E	USE = 1
3:CONTENTS = C	6:LINK = 1,INFO = F	

34.[*29*] 为习题 33 的算法编写一个 MIX 程序,并确定它的运行时间。

35.[*22*] 把这一小节的动态存储分配方法,与在 2.2.2 小节末尾讨论的可变大小的顺序表技术加以对比。

▶ 36.[*20*] 在加利福尼亚的好莱坞,有个便餐店,一排含有 23 个席位。用餐者以一人或二人一伙进入这餐馆,女招待员指示他们在哪里就座。假定任何时候都不会有多于 16 个顾客同时出现,而且如果没有单独来到的顾客被指定到号码为 2,5,8,…,20 中的任何一个席位上,试证明,这位女招待员总能立即让人们入席而不必分开任何一对。(各对一起离开。)

▶ 37.[*26*] 继续习题 36,证明当便餐店仅有 22 个席位时,女招待员不可能总是这样好地进行工作:不管她使用什么策略,总是可能碰到这样一种情况:两个朋友进入餐馆,而且仅有 14 人已入席,但是却没有两个相邻的座位空着。

38.[*M21*] (J.M.Robson)习题 36 和 37 的便餐店问题可以推而广之,用以说明任何不对保留块重新进行再定址的动态存储分配算法的最坏情况特性。命 $N(n,m)$是最小的使得任何要求分配和释放的序列均可不产生溢出地被处理的内存数量,假定所有块的大小都不大于 m 而且所要求的空间总数总不超过 n。习题 36 和 37 证明了 $N(16,2) = 23$;试对于所有的 n,确定 $N(n,2)$的精确值。

39.[*HM23*](J.M.Robson) 在习题 38 的记号下,证明 $N(n_1 + n_2, m) \leqslant N(n_1, m) + N(n_2, m) + N(2m - 2, m)$;因此对于固定的 m,$\lim_{n\to\infty} N(n,m)/n = N(m)$存在。

40.[*HM50*] 继续习题 39,确定 $N(3)$,$N(4)$,以及 $\lim_{m\to\infty} N(m)/\lg m$(如果它存在的话)。

41.[*M27*] 这道习题的目的,是考虑伙伴系统的最坏情况内存用法。例如,会出现这样一个特别坏的情况——如果我们从一个空的内存开始,并如下进行之:首先保留长度为 1 的 $n = 2^{r+1}$ 个块,它从单元 0 开始到 $n - 1$;然后对于 $k = 1,2,\cdots,r$,释放全部其开始单元不可被 2^k 整除的块,并保留 $2^{-k-1}n$ 个长度为 2^k 的块,这些块从单元$\frac{1}{2}(1 + k)n$ 开始到单元$\frac{1}{2}(2 + k)n - 1$ 为止。这个过程竟动用了如此之多的内存,达到曾被占用过内存的 $1 + \frac{1}{2}r$ 倍!

证明最坏的情况实质上不会比以下这种情况更坏：当所有的请求是针对大小为 $1,2,\cdots,2^r$ 的块的，而且如果在任何时候所要求的总空间绝不超过 n，其中 n 是 2^r 的一个倍数，那么，伙伴系统将不会溢出一个大小为 $(r+1)n$ 的内存区域。

42.［*M40*］(J.M.Robson,1975) 当如同在习题 38 中一样对于存储分配使用最好适合方法时，设 $N_{BF}(n,m)$ 是为保证不溢出所需要的内存数量。试找出一个进攻性策略来证明 $N_{BF}(n,m)\geqslant mn-O(n+m^2)$。

43.［*HM35*］继续习题 42，设 $N_{FF}(n,m)$ 是使用首先适合方法所需要的内存数量。试找出一个防御性策略来证明 $N_{FF}(n,m)\leqslant H_m n/\ln 2$。(因此首先适合的最坏情况距离最好可能的最坏情况并不远。)

44.［*M21*］假设分布函数 $F(x)=$(一个块有小于等于 x 的大小的概率)是连续的。例如，如果诸大小是在 a 和 b 之间均匀分布的，则对于 $a\leqslant x\leqslant b$，$F(x)$ 是 $(x-a)/(b-a)$。当我们使用分布适合方法时，试给出表达应建立的头 N 个槽的大小的公式。

2.6 历史和文献

保存在连续内存单元中的信息的线性表和矩形数组,从最早期的存储程序计算机问世以来,就已被广泛地应用了,而且最早的关于程序设计的一些论文,就已经给出过遍历这些结构的基本算法。[例如,见 J. von Newmann,*Collected Works* **5** , 113 ~ 116(写于1946 年);M. V. Wilkes, D. J. Wheeler, S. Gill,*The Preparation of Programs for an Electronic Digital Computer*(Reading, Mass.: Addison-Wesley, 1951),子程序 V-1;特别是写于 1945 年的 Konrad Zuse 的著作,*Berichte der Gesellschaft für Mathematik und Datenverarbeitung* **63**(Bonn: 1972)。Zuse 是使用动态可变长度表建立非平凡算法的头一个人。] 在使用变址寄存器之前,顺序线性表的操作,乃是通过对机器语言指令本身实施算术运算完成的,而且这种类型的操作,成为早期追求一台这样的计算机的动机之一,即它的程序与程序所加工的数据,共享内存空间。

允许可变长的线性表,以这样一种方式来共享顺序的单元——即必要时向前或向后移动之,如同在 2.2.2 小节中所描述的那样,这种技术,显然是更晚得多的发明。Digitek 公司的 J. Dunlap 于 1963 年,联系到设计一系列的编译程序,发明了这些技术。大约与此同时,这思想也独立地出现于 IBM 公司的 COBOL 编译程序的设计中,而且一套称为 CITRUS 的有关子程序的汇集,陆续地被使用各种装置中。这些技术一直保密,直到挪威的 Jan Garwick 独立地提出了它们之后才公开;见*BIT* **4** (1964), 137 ~ 140。

存线性表于非顺序单元中的思想,其起源似乎与设计带有转动的磁鼓存储器的计算机有关。这样一种计算机在执行了单元 n 中的指令之后,通常还未做好从单元 $n+1$ 中得到它的下一条指令的准备,因为磁鼓已经转过了这一点。取决于正被实施的指令,下一条指令的最有利的位置,可能是比如说 $n+7$ 或 $n+18$,而且如果机器的指令最佳地而不是连续地来放置,则这种机器的操作速度可提高六到七倍。[至于关于这些指令的最佳位置这一有趣问题的讨论,见作者的论文,*JACM* **8** (1961), 119 ~ 150。] 因此,在每条机器语言指令中提供了一个额外的地址字段,作为指向下一条指令的链接。这种思想,叫做"一加一寻址",是 J. Mauchly 于 1946 年讨论的[*Theory and Techniques for the Design of Electronic Computers* **4** (University of Pennsylvania, 1946),讲义 37];它包含了在萌芽状态的链接表的思想,尽管在这一章里我们已经如此频繁地使用的动态插入和删除操作在当时还并不为人所知。早期程序链接的另一个出现,见 H. P. Luhn 提议对外部检索使用"链接"的 1953 年备忘录;参考 6.4 节。

当 A. Newell, J. C. Shaw 和 H. A. Simon 开始他们的"机器启发式问题解决"研究时,链式存储技术就真正地产生了。为编写寻求数理逻辑证明的程序,作为一项辅助工具,他们于 1956 年春天设计了头一个列表处理语言 IPL-II。(IPL 表示 Information Processing Language,信息处理语言。)这一系统使用了指针并且包括了诸如可用空间表这样一些重要概念,但是,当时栈的概念尚未完善地形成。IPL-III 是在一年之后设计的,它包含了栈的"下推"和"弹出",作为重要的基本操作。[IPL-II 文献见 *IRE Transactions* **IT-2** (1956 年 9 月), 61 ~ 70; *Proc. Western Joint Comp. Conf.* **9** (1957), 218 ~ 240。关于 IPL

-III 的材料，第一次出现于 1957 年夏天密执安大学的课程讲义中。]

Newell，Shaw 和 Simon 的工作引发了许多人使用链接存储（当时这叫做 NSS 存储），但主要用于模拟人类思维的过程。逐渐地这些技术才被确认为计算机程序设计的基本工具；描述链式存储对于"实实在在"问题之有用性的第一篇论文，由 J. W. Carr，III 发表于*CACM* **2**，2（1959 年 2 月），4～6 上。Carr 在这篇论文中指出，在通常的程序设计语言中，可以很容易地处理链接表，而无须要求有灵巧的子程序或解释系统。还可看 G. A. Blaauw 的文章，"Indexing and control-word techniques"，*IBM J. Res. and Dev*. **3**（1959），288～301。

开始是在链接表中使用一个字的节点，但是大约 1959 年，一些不同团体的人们，逐渐发现了每个节点有若干个连续的字和"多重链接"表的有用价值。专门讨论这一思想的头一篇论文，是 D. T. Ross 发表的，*CACM* **4**（1961），147～150。当时他对于在这一章中称做节点（node）的对象，使用了"丛（plex）"这个词，但他随后又在不同的意义下使用 plex 这一词，来表示与有关的遍历它们的算法相结合的一个节点类。

为访问节点内的字段，所用的记号一般有两种：字段的名称或者是在指针的名称之前，或者在其之后。于是，尽管我们在这一章中已经写成"INFO(P)"，另一些作者却写成"P.INFO"。在准备编写这一章时，这两种记号似乎同样都是突出的。但是，这里采用的记号更有其优越性，这就是，如果我们定义 INFO 和 LINK 数组，并且使用 P 作为下标，则它就立即可翻译成 FORTRAN，COBOL 式类似的语言。况且使用数学函数记号来描述一个节点的属性更显得自然些。注意"INFO(P)"在通常的数学表示中，读作"info of P"，恰如 $f(x)$ 被读作"f of x"。而另外一种记法 P.INFO，因为它趋向于强调 P，尽管它可以读作"P's info"，但却缺乏自然的风味；INFO(P)显得更可取的原因，显然在于这样一个事实，即当使用此记号时，P 是变量，但 INFO 则有着一个固定的意义。用类推的方法，我们可以把一个向量 $A=(A[1],A[2],\cdots,A[100])$ 考虑成为一个有 100 个名为 1，2，…，100 的字段的节点。现在在我们的记号下，第 2 个字段将作为"2(P)"来访问，其中 P 指向向量 A；但如果我们正在访问的是该向量的第 j 个元素，则我们发现把变量"j"写在第二位，写成 $A[j]$ 更为自然些。类似地，把可变的量"P"写在记号 INFO(P)中的第二位，似乎也更为适当。

也许，第一个认识到"栈"（后进先出）和"队列"（先进先出）概念是重要研究对象的人，是关注于降低所得税的成本会计员；关于存货计价的"LIFO"和"FIFO"方法的讨论，可参看任何中等会计教程。例如 C. F. Schlatter 和 W. J. Schlatter 的*Cost Accounting*（New York：Wiley，1957），第 7 章。20 世纪 40 年代中期，A. M. Turing 提出了用于子程序链接、局部变量和参数的称为逆转存储的栈机制。他将"push（推）"和"pop（弹）"命名为"bury（埋）"和"disinter/unbury（掘/挖）（见 1.4.5 小节的参考文献）。毫无疑问，在计算机程序设计中把栈保持在顺序内存单元中的简单用法很早就已经很普遍了，因为栈是一种简单和直观的概念。如上所述，链接形式的栈程序设计，首先出现于 IPL 中；栈这一名称是从 IPL 的术语中引出的（尽管"下推表"在 IPL 中是更正式的词汇），它也是由 E. W. Dijkstra 独立地引进的[*Numer. Math*. **2**（1960），312～318]。"双端队列"一词是由 E. J. Schweppe 于 1966 年创造的。

循环的和双重链接的表,其起源说不清楚;也许这些思想对于许多人来说是自然出现的。一种促使这些技术普及的强烈因素,是以它们为基础的一般列表处理系统的存在[主要是 J. Weizenbaum 的打结列表结构(Knotted List Structures), *CACM* **5** (1962),161 ~ 165 以及对称列表处理程序(Symmetric List Processor), *CACM* **6** (1963),524 ~ 544]。Ivan Sutherland 在他的草图(Sketchpad)系统中介绍了在更大的节点内独立双重链接表的使用(麻省理工学院博士论文,1963 年)。

自从早期的计算机出现以来,有才干的程序员们已经各自开发了种种编址和遍历多维信息数组的方法,从而就出现了另一部分未发表的民间计算机创作。这一课题是由 H. Hellerman 首先加以书面综述的,见*CACM* **5** (1962),205 ~ 207。也可见 J. C. Gower, *Comp. J.* **4** (1962),280 ~ 286。

明显地表示在计算机内存中的树结构,开始是用来处理代数公式的。若干早期的计算机的机器语言使用三地址代码来表示算术表达式的计算;这三地址代码就等价于一个二叉树表示的 INFO,LLINK 和 RLINK。1952 年,H. G. Kahrimanian 建立了求以扩充的三地址代码表示的代数公式之微分的算法;见 *Symposium on Automatic Programming* (Washington,D.C.:Office of Naval Research,1954 年 5 月),6 ~ 14。

从此以后,各种外观的树结构,同大量的计算机应用相联系,已由许多人独立地进行了研究;但是,树操作的基本技术(不是一般的列表操作)除了具体算法的详细说明之外,很少出现在出版物中。K. E. Iverson 和 L. R. Johnson,联系到对所有数据结构更一般的研究,第一次作出了综述[IBM 公司研究报告 RC-390,RC-603,1961;见 Iverson, *A Programming Language* (New York:Wiley,1962),第 3 章]。也见 G. Salton,*CACM* **5** (1962),103 ~ 114。

穿线树的概念是由 A. J. Perlis 和 C. Thornton 给出的,*CACM* **3** (1960),195 ~ 204。他们的文章也介绍了在各种顺序下遍历树的重要思想,并给出了代数操作算法的大量例子。遗憾的是,这一篇重要的文章是匆忙抛出的,因此它包含许多印刷错误。Perlis 和 Thornton 的穿线树,按照我们的术语实质上只是“右穿线树”;在两个方向下都穿线的二叉树,是由 A. W. Holt 独立地发现的,见*A Mathematical and Applied Investigation of Tree Structures* (宾夕法尼亚大学博士论文,1963)。树节点的后根序和先根序,被 Z. Pawlak 称为“正常进行的次序”和“双向进行的次序”[*Colloquium on the Foundation of Mathematics*, Tihany,1962 (Budapest:Akadémiai Kiadó,1965),227 ~ 238]。在上面引证的 Iverson 和 Johnson 的文献中,先根序又称为“子树序”。A. G. Oettinger 描述了表示树结构与对应的线性记号之间联系的图形方式,*Proc. Harvard Symp. on Digital Computers and their Applications* (1961 年 4 月),203 ~ 224。S. Gorn 介绍了通过度数在先根序下表示树,以及把这个表示与杜威十进记号以及树的其它性质加以联系的有关算法,见*Proc. Symp. Math. Theory of Automata* (Brooklyn:Poly. Inst.,1962),223 ~ 240。

在 2.3.4.6 小节中,回顾了树结构作为数学对象的历史,并给出了这一课题的参考文献。

当在 1966 年最初编写这一节时,关于信息结构最普遍的知识,乃是来自程序员对列表处理系统的揭示——列表处理系统在这段历史中有着非常重要的地位。第一个广

泛使用的系统是 IPL-V(IPL-III 的后继,迟至 1959 年才开发出来);IPL-V 是一个解释系统,程序员从中学到了一种关于列表操作的类机器语言。同时,FLPL(用于列表处理的一组 FORTRAN 子程序,它也是受到 IPL 的启发,但使用子程序调用以代替解释语言)为 H. Gelernter 和其他人所开发。第三个系统 LISP,为 J. McCarthy 所设计,也是在 1959 年。LISP 截然不同于它的先驱者:它的程序过去都是(现在仍然是)表达成与“条件表达式”相结合的数学函数记号,然后变换成一种列表表示。在 20 世纪 60 年代期间,已有许多列表处理系统问世;它们当中在历史上最突出的是 J.Weizenbaum 的 SLIP,这是一组以双重链接列表进行操作的,用于 FORTRAN 程序的子程序。

Bobrow 和 Raphael 的一篇论文,*CACM* **7** (1964),231 ~ 240,可以作为对 IPL-V,LISP 和 SLIP 的一个简短介绍;它还给出了对这些系统的比较。P. M. Woodward 和 D. P. Jenkins 发表了对 LISP 的精采介绍,*Comp.J.* **4** (1961),47 ~ 53。也请见作者们对他们自己的系统的讨论,他们的每一篇论文都有着重要的历史意义:A. Newell 和 F. M. Tonge,“An introduction to IPL-V”,*CACM* **3** (1960),205 ~ 211;H. Gelernter,J. R. Hansen 和 C. L. Gerberich,“A FORTRAN-compiled List Processing Language”,*JACM* **7** (1960),87 ~ 101;John McCarthy,“Recursive functions of symbolic expressions and their computation by machine”, *CACM* **3** (1960),184 ~ 195;J. Weizenbaum,“Symmetric List Processor”,*CACM* **6** (1963),524 ~ 544。后一篇论文包括用于 SLIP 中的所有算法的完备描述。在所有这些早期的系统中,仅仅 LISP 含有有用成分而在接着而来数十年的进步中保留下来。McCarthy 描述了 LISP 的早期历史,*History of Programming Languages* (Academic Press,1981),173 ~ 197。

20 世纪 60 年代还出现了若干字符串处理系统;这些系统主要涉及字符信息的可变长字符串的运算——寻找某些子串和用其它子串替换,等等。其中,以历史的角度看最重要的是 COMIT[V. H. Yngve, *CACM* **6** (1963),83 ~ 84]和 SNOBOL[D. J. Farber,R. E. Griswold 和 I. P. Polonsky, *JACM* **11** (1964),21 ~ 30]。尽管字符串处理系统已得到广泛使用,而且它们主要是由我们在这一章中所见到的一些算法组成,但是,它们在信息结构表示技术的历史上,所起的作用相对来说比较小;这些系统的用户已经和实际的计算机内部实现过程的细节相隔绝。关于字符串操作技术的概述,见 S. E. Madnick,*CACM* **10** (1967),420 ~ 424。

关于列表处理的 IPL-V 和 FLPL 系统对于共享列表问题而言既不使用废料收集,也不使用访问计数技术;而是,每个列表为一个列表“所固有”,又为访问这一列表的任何其它列表“所借用”,而且当一个列表的“持有者”允许时,可删去一个列表。因此,程序员必须确保不得有列表借用任何正被删去的列表。关于列表的访问计数器技术,是由 G. E. Collins 引入的,*CACM* **3** (1960),655 ~ 657;在*CACM* **9** (1966),578 ~ 588 上,他又对它作了进一步的说明。废料收集第一次描述于 McCarthy 1960 年的一篇文章中;另见在 *CACM* **7** (1964),38 上 Weizenbaum 的说明,以及 Cohen 和 Trilling 的一篇论文,*BIT* **7** (1967),22 ~ 30。

对链接处理重要性日益增长的认识,自然导致它们侵入到 1965 年以后设计的代数程序设计语言中。新的语言允许程序员选择适当的数据表示形式,而不必乞怜于汇编语言或付出通用列表结构的开销。在这一发展中的某些基本行程, 有这样一些人的工

作:N. Wirth 和 H. Weber [*CACM* **9** (1966),13 ~ 23,25,89 ~ 99];H. W. Lawson [*CACM* **10** (1967),358 ~ 367];C. A. R. Hoare [*Symbol Manipulation Languages and Techniques*, D. G. Bobrow 编辑,(Amsterdam: North Holland, 1968), 262 ~ 284], O.-J. Dahl 和 K. Nygaard [*CACM* **9** (1966),671 ~ 678];A. van Wijingaarden, B. J. Mailloux, J. E. L. Peck 和 C. H. A. Koster[*Numerische Math*. **14** (1969),79 ~ 218];Dennis M. Ritchie [*History of Programming Language-II* (ACM Press,1996),671 ~ 698]。

动态存储分配算法在见诸文字描述之前,就已经被使用若干年了。W. T. Comfort 于 1961 年准备了一篇颇具可读性的论述,发表于*CACM* **7** (1964),357 ~ 362。在 2.5 节中所介绍的“边界标记”方法,是作者于 1962 年设计的,用于 Burroughs B5000 计算机的操作系统中。“伙伴系统”首先是由 H. Markowitz 同 SIMSCRIPT 程序设计系统,一起于 1963 年使用的,而且 K. Knowlton 也独立地发现并发表之,*CACM* **8** (1965),623 ~ 625;也见 *CACM* **9** (1966),616 ~ 625。关于动态存储分配的进一步讨论,见 Iliffe 和 Jodeit 的论文,*Comp. J.* **5** (1962),200 ~ 209;Bailey, Barnett 和 Burleson,*CACM* **7** (1964),339 ~ 346;A. T. Berztiss,*CACM* **8** (1965),512 ~ 513;以及 D. T. Ross, *CACM* **10** (1967),481 ~ 492。

Mary d'Imperio 的“Data Structures and their Representaton in Storage”,*Annual Review in Automatic Programming* **5** (Oxford: Pergamon Press, 1969),给出了关于信息结构与程序设计的一般关系的讨论。这篇论文,也是对于这一课题历史的有价值的指南,因为它包括了联系到 12 个列表处理和字符串处理系统所用结构的详细分析。也可见两个讨论会的论文集,*CACM* **3** (1960),183 ~ 234 和*CACM* **9** (1966),567 ~ 643,以了解更深入的历史细节(上面已经引用了这些论文集中若干独立的文章)。

Jean E. Sammet 编辑了一份带评注的有关符号处理和代数公式处理早期工作的优秀书目,与本章内容有极大联系,见*Computing Reviews* **7** (1966 年 7 月—8 月),B1 ~ B31。

在这一章中,我们已详尽地考察了信息结构的各种具体类型,而且(为免得我们见树不见林),还清点了一下我们所学过的内容,并从一个更宽阔的视野来简单概括关于信息结构的一般课题,这也许是明智的。从节点作为的元素的基本思想开始,我们已经见到许多例子,它们都说明了表示结构关系的方便的方法——或者是隐含地(以节点在计算机内存中存放的相对次序为基础)或者是明显地(借助于节点中指向其它节点之链接)。应当在计算机程序的表格之内予以表示的结构信息的数量,取决于对这些节点要实施的操作。

由于教学法上的原因,我们已经在很大程度上专注在信息结构与它们的机器表示之间的联系,而不是孤立地来讨论这些问题。然而,为了更深入的理解,从更抽象的观点,即“抽取”通对它们本身就可加以研究的若干层次的思想,来讨论这一课题是有帮助的。已经提出了这类值得注意的若干方法,从早期的论著中我们特别推荐下列思想深邃的论文:G. Mealy,“Another look at data”, *Proc. AFIPS Fall Joint. Computer Conf.* **31** (1967), 525 ~ 534; J. Earley,“Toward an understanding of data structures”,*CACM* **14** (1971),617 ~ 627;A. R. Hoare,“Notes on data structuring”,见于 O.-J. Dahl, E. W. Dijkstra 和 C. A. R. Hoare,*Structured Programming* (Academic Press, 1972),83 ~ 174;Robert W. Engles,“A tutorial on data-base organization”,*Annual. Review in Automatic Programming* **7** (1972),3 ~ 63。

这一章中的讨论,并未以完全的一般性来概括信息结构的整个课题;至少有三个重要方面,在这里未曾涉及到:

a)通常需要查遍一个表格,以求出具有某个值的一个或一组节点,而对于这样的一项操作的需要通常对于这个表格的结构有着深远的影响。这一情况将留待第 6 章详细剖析。

b)我们主要关注了计算机内部结构表示;但这显然仅仅是事情的一个方面,因为结构还必须表示成外部的输入和输出数据。在简单的情况下,外部结构实质上能以我们所曾经考虑过的同样的技术来处理;但字符串与更复杂结构之间的转换过程,也是非常重要的。这些过程要在第 9 章和第 10 章分析。

c)我们主要讨论了结构在一个高速随机存取内存中的表示。当使用缓慢的存储设备例如磁盘或磁带时,我们发现所有的结构问题,都变得更为迫切;数据表示的有效算法和有效方案变得更为关键。在这样一些情况下彼此相互链接的节点应当进入到存储的附近区域。通常这些问题都高度依赖于个别机器的特征,所以难以一般地进行讨论。这一章所处理的较简单的例子,应当对读者解决与不大理想的存储设备有关的更困难的问题有所帮助;第 5 章和第 6 章详细地讨论这些问题。

这一章处理的课题中主要的含意是什么? 也许,我们所能作出的最重要的结论是:我们所触及的思想不仅限于计算机程序设计,它们更一般地可应用于日常生活中。一组包含着一些字段的节点,其中某些字段指向其它的节点,对于各类结构关系来说,看来是一个非常好的抽象模型。这一模型说明,我们可如何从一些简单的结构来构造出复杂的结构,而且我们已经看到,用于处理这一结构的相应的算法,可以以一种自然的方式来设计。

因此,提出比我们现时所知道的更多的关于节点链接集合的理论,似乎是适当的。也许创始这样一套理论的最明显的途径,就是定义新的一类处理链接结构的抽象机器或“自动机”。例如,这样一种自动机可以非正式地定义如下:有数 k, l, r 和 s,使得自动机处理含有 k 个链接字段和 r 个信息字段的节点;它有 l 个链接寄存器和 s 个信息寄存器,这使它得以控制它所实现的过程。信息字段和寄存器可以包含来自信息符号的某个给定集合中的任何符号;每个链接字段和链接寄存器或者包含 Λ,或者指向一个节点。机器能够(i)建立新的节点(置链到该节点的链接到一个寄存器中),(ii)比较诸信息符号或链接值是否相等,(iii)在寄存器与节点之间传送信息符号或链接值。仅仅是由链接寄存器指向的节点,才是可直接存取的。对机器行为的适当限制,将使它等价于若干其它种类的自动机。

早在 1952 年 A. N. Kolmogorov 即已提出了一个相关的计算模型。他的机器基本上在图 G 上操作,图 G 有一个特别指定的开始向量 v_0。每一步动作仅依赖于由 G 中距 v_0 小于等于 n 的所有向量组成的图 G',在 G 中用另一个图 $G''=f(G')$ 代替 G',其中 G'' 包含 v_0 和距 v_0 为 n 的向量,以及可能的其它向量(新建立的向量);图 G 的剩余部分保留不变。这里 n 是为任一特定算法事先指定的固定值,但可以任意大。每个向量附上有限字符集中的符号,并限制使得没有两个同符号向量连接到同一个公共向量上。(见 A. N. Kolmogorov, *Uspekhi Mat. Nauk* **8**,4(1953),175～176;Kolmogorov 和 Uspensky, *Uspèkhi Mat. Nank*

13,4 (1958),3~28;*Amer. Math. Soc. Translations*,Series 2,**29** (1963),217~245。)

链接自动机可以很容易地模拟图形机,每一图形步至多执行有限的步而已。反之,图形机不可能模拟任意链接自动机而不极大地增加运行时间,除非由于对向量度数的限制,定义从无向图改为有向图。当然链接模型非常接近于实际机器上程序员可用的操作,而图形模型则不然。

对于这样的设备需要解决的某些最有趣的问题将是,确定它们能多快地解决某些问题,或者它们需要多少节点以解决某些问题(例如,为翻译某种形式语言)。在写完这一章时,关于这方面的若干有趣的结果,已经(特别是由 J. Hartmanis 和 R. E. Stearns)得到了,但仅对具有多重带和读/写头等的特殊类型的图灵机有效;由于图灵机的模型相对来说是不现实的,这些结果对于实际问题一般用处不大。

我们必须承认,为一个链接自动机所建立的节点的个数 n 趋向无限时,我们不知道怎样物理地构造这样一个设备,因为我们希望,不管 n 的大小如何,该机器的各种操作将要花费同样数量的时间;如果像在计算机的内存中那样,通过使用地址表示链接,则有必要对节点个数设置一个界限,因为链接字段有一个固定的大小。因此当 n 趋向无限时,多带图灵机是一个更为现实的模型。然而似乎有理由相信,如上所述的一个链接自动机,会导致比图灵机更适当的算法复杂性理论,甚至当考虑对于充分大的 n 的渐近公式时也是如此,因为该理论更像是与实际的 n 值相关的。而且当 n 变成大于 10^{30}左右时,没有哪台一条带的图灵机会是实际的:它绝对构造不出来。中肯比现实主义要更为重要。

自作者写上述的评论的大部分以来,已经过了许多年了,每个人都会因为链接自动机(现在叫做指针机器)理论确实取得的很大进步而感到高兴。但是当然还有许多事情有待完成。

General rules for programming have been disovered.
Most of them have been used in the
Kansas City freight yards for a long time.
程序设计的一般规则已被发现了,
长久以来,它们的大多数都已被用
于堪萨斯市的货运调度场中
—Derrick Lehmer (1949)

You will, I am sure, agree with me…that if page
534 finds us only in the second chapter, the length of
the first one must have been really intolerable.
我确信,你将和我取得共识…即如果
到 534 页我们仍在第 2 章中,则第 1 章
确实太长了。
Sherlock Holmes, *The valley of Fear* (1888)

习题答案

I am not bound to please thee with my answers.
我的回答本来就不是为要讨你喜欢。
Shylock, in *The Merchant of Venice* (Act IV, Scene 1,Line 65)

关于习题的说明

1. 对于擅长数学的读者,这是一个普通的问题。

4. 见 W. J. LeVeque,*Topics in Number Theory* **2** (Reading, Mass.: Addison-Wesley,1956),第 3 章;P. Ribenboim,*13 Lectures on Fermat's Theorem* (New York: Springer-Verlag, 1979); A. Wiles, *Annals of Mathematics* **141** (1995),443~451。

1.1 节

1. $t \leftarrow a, a \leftarrow b, b \leftarrow c, c \leftarrow d, d \leftarrow t$。

2. 第一次之后,变量 m 和 n 之值分别是以前的 n 和 r 之值;而且 $n > r$。

3. **算法 F**(欧几里得算法) 给定两个正整数 m 和 n,求它们的最大公因子。

F1.[余数 m/n] 以 n 除 m 并令 m 是余数。
F2.[它为 0?] 如果 $m=0$,则此算法以 n 为答案而终止。
F3.[余数 n/m] 以 m 除 n 并令 n 是余数。
F4.[它为 0?] 如果 $n=0$,则算法以答案 m 而终止;否则返回步骤 F1。▌

4. 由算法 E,$n = 6099, 2166, 1767, 399, 171, 57$。答案:57。

5. 既不是有限的,也不是确定的,也不是有效的,还可能无输出;就格式而言,在步骤号之前没有字母,不出现有概括的短语,而且没有"▌"。

6. 以 $n=5$ 和 $m=1,2,3,4,5$ 来试验算法 E,我们发现步骤 E1 被执行的次数分别为 2,3,4,3,1 次;因此平均是 $2.6 = T_5$。

7. 除了个别情况下,$n > m$。而且当 $n > m$ 时,算法 E 的第一次迭代仅仅是互换这两个数,所以 $U_m = T_m + 1$。

8. 命 $A = \{a, b, c\}$, $N = 5$。这个算法将以字符串 $a^{\gcd(m,n)}$ 结束。

j	θ_j	ϕ_j	b_j	a_j	
0	ab	(空)	1	2	删掉一个 a 和一个 b,或转到 2
1	(空)	c	0	0	把 c 加到左端,返回到 0
2	a	b	2	3	把所有 a 变成 b
3	c	a	3	4	把所有 c 变成 a
4	b	b	0	5	如果还剩有 b,则重复

9. 例如,我们可以说 C_2 表示 C_1,如果有从 I_1 到 I_2 的函数 g,从 Q_2 到 Q_1 的函数 h,和从 Q_2 到正整数的函数 j,满足下列条件:

a) 如果 x 在 I_1 中,则 $h(g(x)) = x$。

b) 如果 q 在 Q_2 中,则 $f_1(h(q)) = h(f_2^{[j(q)]}(q))$,其中 $f_2^{[j(q)]}$ 指的是函数 f_2 被迭代 $j(q)$ 次。

c) 如果 q 在 Q_2 中,则当且仅当 q 在 Ω_2 中时 $h(q)$在 Ω_1 中。

例如,令 C_1 如同(2)中那样,并令 C_2 有 $I_2 = \{(m,n)\}$,$\Omega_2 = \{(m,n,d)\}$,$Q_2 = I_2 \cup \Omega_2 \cup \{(m,n,a,b,1)\} \cup \{(m,n,a,b,r,2)\} \cup \{(m,n,a,b,r,3)\} \cup \{(m,n,a,b,r,4)\} \cup \{(m,n,a,b,5)\}$。令 $f_2((m,n)) = (m,n,n,n,1)$; $f_2((m,n,d)) = (m,n,d)$; $f_2((m,n,a,b,1)) = (m,n,a,b,a \bmod b,2)$; 如果 $r = 0$,$f_2((m,n,a,b,r,2)) = (m,n,b)$,否则 $f_2((m,n,a,b,r,2)) = (m,n,a,b,r,3)$; $f_2((m,n,a,b,r,3)) = (m,n,b,b,r,4)$; $f_2((m,n,a,b,r,4)) = (m,n,a,r,5)$; $f_2((m,n,a,b,5)) = f_2((m,n,a,b,1))$。

现在令 $h((m,n)) = g(m,n) = (m,n)$;$h((m,n,d)) = (d)$;$h((m,n,a,b,1)) = (a,b,0,1)$;$h((m,n,a,b,r,2)) = (a,b,r,2)$;$h((m,n,a,b,r,3)) = (a,b,r,3)$;$h((m,n,a,b,r,4)) = h(f_2((m,n,a,b,r,4)))$;$h((m,n,a,b,5)) = (a,b,b,1)$;$j((m,n,a,b,r,3)) = j((m,n,a,b,r,4)) = 2$,否则 $j(q) = 1$。则 C_2 表示 C_1。

注:试图以一种更简单的方式定义诸事项是令人感兴趣的——例如,令 g 把 Q_1 映像到 Q_2 中,并且坚持仅当 $x_0, x_1, \cdots$ 是 C_1 中的计算序列时,$g(x_0), g(x_1), \cdots$ 是 C_2 中以 $g(x_0)$开始的计算序列的子序列。但这并不充分;在上边的例子中,C_1 忘记了 m 和 n 原来的值,但 C_2 则不然。

如果 C_2 借助于函数 g,h,j 来表示 C_1,而且如果 C_3 借助于函数 g',h',j' 来表示 C_2,则 C_3 借助于函数 g'',h'',j''来表示 C_1,其中

$$g''(x) = g'(g(x)), h''(x) = h(h'(x))$$

而且

$$j''(q) = \sum_{0 \leqslant k < j(h'(q))} j'(q_k)$$

如果 $q_0 = q$,且 $q_{k+1} = f_3^{[j'(q_k)]}(q_k)$。因此上面定义的关系是传递性的。如果函数 j 是有界的,我们可以说 C_2 *直接地表示* C_1;这个关系也是传递的。关系“C_2 表示 C_1”产生一个等价关系,其中,两个计算方法显然地是等价的,当且仅当它们计算其输入之同构函数;关系“C_2 直接地表示 C_1”生成更有趣的等价关系,它也许吻合于是“实质上相同的算法”的直观想法。

关于模拟的另一种方法,请见 R. W. Floyd 和 R. Beigel,*The Language of Machines* (Computer Science Press,1994),第 3.3 节。

1.2.1 小节

1.(a) 证明 $P(0)$。(b) 证明 $P(0), \cdots, P(n)$蕴涵 $P(n+1)$,对所有的 $n \geqslant 0$。

2. 定理对于 $n = 2$ 未曾被证明;在证明的第二部分中,取 $n = 1$;我们在那里假定 $a^{-1} = 1$。如果这个条件为真(即使得 $a = 1$),则定理确实成立。

3. 正确的答案是 $1 - 1/n$。错误出现在对 $n = 1$ 的证明中,这时左边的公式,或者可以假定是无意义的,或者可以假定是 0 (因为有 $n - 1$ 项)。

5. 如果 n 为素数,它不过是素数的乘积。否则 n 有因子,所以对于 $1 < k, m < n$ 的某个 k 和 m,$n = km$。由于 k 和 m 两者小于 n,由归纳法它们可以写作素数的乘积;因此 n 是出现在 k 和 m 的表示中素数的乘积。

6. 在图 4 的记号下,我们证明,A5 意味着 A6。这是显然的,因为 A5 意味着$(a' - qa)m + (b' - qb)n = (a'm + b'n) - q(am + bn) = c - qd = r$。

7. $n^2-(n-1)^2+\cdots-(-1)^n1^2=1+2+\cdots+n=n(n+1)/2$。

8. (a)我们将证明$(n^2-n+1)+(n^2-n+3)+\cdots+(n^2+n-1)$等于 n^3。这个和是$(1+3+\cdots+(n^2+n-1))-(1+3+\cdots+(n^2-n-1))=((n^2+n)/2)^2-((n^2-n)/2)^2=n^3$。我们已经使用了等式(2)；然而，要求的是一个归纳证明，所以应该采用另一种方法！对于 $n=1$，结果是明显的。设 $n\geqslant1$；$(n+1)^2-(n+1)=n^2-n+2n$，所以对于 $n+1$，前头诸项都大 $2n$；于是对于 $n+1$，所求和是对于 n 的和加上$\underbrace{2n+\cdots+2n}_{n次}+(n+1)^2+(n+1)-1$；这就等于 $n^3+2n^2+n^2+3n+1=(n+1)^3$。(b) 我们已经看到，对于$(n+1)^3$ 的第一项比对于 n^3 的最后一项大 2。因此由等式(2)，$1^3+2^3+\cdots+n^3=$由 1 开始的连续的奇数之和$=$(项数)$^2=(1+2+\cdots+n)^2$。

10. 对于 $n=10$ 是显然的；如果 $n\geqslant10$，则我们有 $2^{n+1}=2\cdot2^n>(1+1/n)^32^n$，而且由归纳法，这就大于$(1+1/n)^3n^3=(n+1)^3$。

11. $(-1)^n(n+1)/(4(n+1)^2+1)$。

12. 这一推广的惟一不平凡部分，是计算 E2 中整数 q。这可以通过反复的减法，把问题归结成确定 $u+v\sqrt{2}$是正的，负的或是零，而这后一个问题易于解决。

不难证明，每当 $u+v\sqrt{2}=u'+v'\sqrt{2}$时，我们必有 $u=u'$ 和 $v=v'$，因为$\sqrt{2}$是无理数。现在很清楚，1 与$\sqrt{2}$没有公因子，如果我们如下定义因子，即 $u+v\sqrt{2}$整除 $a(u+v\sqrt{2})$，当且仅当 a 是一个整数。以这种方式推广的算法计算它的输入之比的正则连分数；参见 4.5.3 小节。

[注：然而，如果我们把因子的概念推而广之：就是说 $u+v\sqrt{2}$整除 $a(u+v\sqrt{2})$，当且仅当对于整数 u'和v'，a 有 $u'+v'\sqrt{2}$的形式，那么，就有一种方法来推广算法 E，使得它总会终止：如果在步骤 E2 中有 $c=u+v\sqrt{2}$和 $d=u'+v'\sqrt{2}$，计算 $c/d=c(u'-v'\sqrt{2})/(u'^2-2v'^2)=x+y\sqrt{2}$，其中 x 和 y 是有理数。现在命 $q=u''+v''\sqrt{2}$，其中 u''和 v''是最接近 x 和 y 的整数；并命 $r=c-dd$。如果 $r=u'''+v'''\sqrt{2}$，从而得出 $|u'''^2-2v'''^2|<|u'^2-2v'^2|$，则计算将终止。关于进一步的介绍，请参看数论教科书中的“二次欧几里得域”。]

13. 把“$T<3(n-d)+k$”加到 $A3,A4,A5,A6$ 的断言中，其中 k 分别取 2,3,3,1 的值。同时也把“$d>0$”加到 $A4$ 中。

15. a)在 iii)中命 $A=S$；每一个非空的良序集合有一个最小的元素。

b)命 $x\prec y$，如果$|x|<|y|$，或者如果$|x|=|y|$且 $x<0<y$。

c)否，所有正实数的子集不满足 iii)。[注：使用所谓的选择公量，可以给出较为复杂的论证，以证明每个集合无论如何总能成为良序的；但是尚没有人能定义使实数成为良序的明显的关系。]

d)为对于 T_n 证明 iii)，对 n 用归纳法：设 A 是 T_n 的非空子集并考虑 A_1，即 A 的第一个分量的集合。由于 A_1 是 S 的非空子集，而且 S 是良序的，A_1 含有一个最小的元素 x。现在考虑 A_x，即 A 的子集，其中之第一个分量等于 x；A_x 可以考虑作 T_{n-1}的子集，如果它的第一个分量被隐去的话，所以由归纳法，A_x 包含一个最小的元素$(x,x_2,\cdots,x_n)$，事实上它就是 A 的最小元素。

e)否，尽管性质 i)和 ii)都成立。如果 S 至少含有两个不同的元素，$a\prec b$，则集合(b)，(a,b)，(a,a,b)，(a,a,a,b)，(a,a,a,a,b)，…没有最小的元素。另一方面，T 可以是良序的，如果我们在 T_n 中定义$(x_1,\cdots,x_m)\prec(y_1,\cdots,y_n)$每当 $m<n$，或者 $m=n$ 且$(x_1,\cdots,x_n)\prec(y_1,\cdots,y_n)$的话。

f)设 S 通过$\prec$而良序化。如果存在这样的无穷序列，则由该序列之成员所组成的集合 A 不

满足性质 iii),因为在该序列中没有元素能是最小的。反之,如果 $<$ 是一个满足 i)和 ii)但不满足 iii)的关系,设 A 是 S 的没有最小元素的非空子集。由于 A 非空,我们可以在 A 中找到 x_1;由于 x_1 不是 A 的最小元素,故在 A 中有 x_2 使得 $x_2 < x_1$;由于 x_2 也不是最小的元素,我们可以找到 $x_3 < x_2$,等等。

g)令 A 是使得 $P(x)$ 为假的所有的 x 的集合。如果 A 非空,则它含有最小的元素 x_0。因此 $P(y)$ 对于所有 $y < x_0$ 为真。但这意味着 $P(x_0)$ 为真,所以 x_0 不在 A 中(矛盾)。因此 A 必须为空:即 $P(x)$ 总是为真。

1.2.2 小节

1. 不存在;因为如果 r 是正的有理数,则 $r/2$ 是更小的正有理数。

2. 不是,如果在一行上出现有无限多个 9 的话;在这种情况下,根据等式(2),此数的十进展开式为 $1 + .24000000\cdots$。

3. $-1/27$,但正文没有定义它。

4. 4。

6. 一个数的十进展开式是惟一的,所以 $x = y$ 当且仅当 $m = n$ 而且对于所有 $i \geq 1$, $d_i = e_i$。如果 $x \neq y$,则人们可以把 m 与 n, d_1 与 e_1, d_2 与 e_2 等等加以比较;当出现第一个不相等时,较大的量即属于 $\{x, y\}$ 中之较大者。

7. 可以对 x 用归纳法,首先对 x 为正证明定律,而后对 x 为负证明。这里把细节略去。

8. 通过对 $n = 0, 1, 2, \cdots$ 进行试验,我们求出使 $n^m \leq u < (n+1)^m$ 的 n 值。归纳地假定 n, $d_1, \cdots, d_{k-1}$ 已经确定,d_k 是使得

$$\left(n + \frac{d_1}{10} + \cdots + \frac{d_k}{10^k}\right)^m \leq u < \left(n + \frac{d_1}{10} + \cdots + \frac{d_k}{10^k} + \frac{1}{10^k}\right)^m$$

的数字。

9. $((b^{p/q})^{u/v})^{qv} = (((b^{p/q})^{u/v})^v)^q = ((b^{p/q})^u)^q = ((b^{p/q})^q)^u = b^{pu}$,因此 $(b^{p/q})^{u/v} = b^{pu/qv}$。这就证明了第二个定律。我们利用第二个来证明第一个定律:$b^{p/q}b^{u/v} = (b^{1/qv})^{pv}(b^{1/qv})^{qu} = (b^{1/qv})^{pv+pu} = b^{p/q+u/v}$。

10. 如果 $\log_{10} 2 = p/q$,其中 p, q 为正,则 $2^q = 10^p$,这是荒谬的,因为右边可为 5 整除,但左边不能。

11. 无限多!不论给出多少位的 x,如果 x 的数字同 $\log_{10} 2$ 的数字一致的话,我们将不知道 $10^x = 1.99999\cdots$ 还是 $2.00000\cdots$。在这一点上,并没有什么神秘或不合理的;如果把 $.444444\cdots$ 加到 $.55555\cdots$,则在加法中就出现类似的情况。

12. 它们是满足等式(7)的 $d_1, \cdots, d_8$ 的仅有的值。

13. (a) 首先用归纳法证明,如果 $y > 0$,则 $1 + ny \leq (1+y)^n$。然后置 $y = x/n$,并取 n 次根。(b) $x = b - 1$, $n = 10^k$。

14. 在(5)的第二个等式中置 $x = \log_b c$,并对该等式的两边取对数。

15. 通过把"$\log_b y$"移到等式的另一边,并且利用(11)来证明之。

16. $\ln x / \ln 10$。

17. 5; 1; 1; 0; 未定义。

18. 否,$\log_8 x = \lg x / \lg 8 = \frac{1}{3} \lg x$。

19. 是,因为 $\lg n < (\log_{10} n)/.301 < 14/.301 < 47$。

20. 它们互为倒数。

21. $(\ln\ln x - \ln\ln b)/\ln b$。

22. 由附录 A 中所附的表, $\lg x \approx 1.442695 \ln x$; $\log_{10} x \approx .4342945 \ln x$。相对误差 $\approx (1.442695 - 1.4342945)/1.442695 \approx 0.582\%$。

23. 取面积 $\ln y$ 的图形,以 x 除它的高度同时以 x 乘它的长度。这个变形保持其面积,并使得它全等于左边的那块——当从 $\ln xy$ 中消去 $\ln x$ 时,因为在对于 $\ln xy$ 的图示中,于点 $x + xt$ 处的高度为 $1/(x + kt) = (1/(1 + t))/x$。

24. 在出现 10 的任何地方代之以 2。

25. 注意,当 p 是精确度(在小数点之后二进制数字的个数)时, $z = 2^{-p}\lfloor 2^{p-k}x\rfloor > 0$。量 $y + \log_b x$ 仍近似地保持为常数。

27. 对 k 用归纳法,可证明

$$x^{2^k}(1-\delta)^{2^{k+1}-1} \leqslant 10^{2^k(n+b_1/2+\cdots+b_k/2^k)}x_k' \leqslant x^{2^k}(1+\epsilon)^{2^{k+1}-1}$$

并取对数。

28. 下面的解使用了与以前一样的辅助表格。

E1. [初始化] 如果 $1-\epsilon$ 是 x 的最大可能值,则置 $y \leftarrow$(最接近 $b^{1-\epsilon}$ 的数), $x \leftarrow 1 - \epsilon - x$, $k \leftarrow 1$。(量 yb^{-x} 在以下步骤中近似地保持为常数。)

E2. [测试是否结束] 如果 $x = 0$,则停止。

E3. [比较] 如果 $x < \log_b(2^k/(2^k-1))$,则将 k 加 1 并重复本步。

E4. [减小数值] 置 $x \leftarrow x - \log_b(2^k/(2^k-1))$, $y \leftarrow y -$(y 向右移 k 位),并转到 E2。▌

如果在步骤 E1 中把 y 置为 $b^{1-\epsilon}(1+\epsilon_0)$,在第 j 次执行步骤 E4 期间,当我们置 $x \leftarrow x + \log_b(1-2^{-k}) + \delta_j$ 和 $y \leftarrow y(1-2^{-k})(1+\epsilon_j)$ 时,对于某个小的误差 δ_j 和 ϵ_j,就引起了计算误差。当算法终止时,我们已经计算了 $y = b^{x-\sum\delta_j}\prod_j(1+\epsilon_j)$。进一步的分析依赖于 b 和计算机的字长。注意在这种情况和习题 26 的两种情况下,如果底数为 e,则有可能进一步改进误差估计,因为对于 k 的大多数值,表项 $\ln(2^k/(2^k-1))$ 可以以高精度给出来:它等于 $2^{-k} + \frac{1}{2}2^{-2k} + \frac{1}{3}2^{-3k} + \cdots$。

注:对于三角函数,可以给出类似的算法;参见 J. E. Meggitt, *IBM J. Res. and Dev.* **6** (1962), 210 ~ 226; **7** (1963), 237 ~ 245。也请见 T. C. Chen, *IBM J. Res. and Dev.* **16** (1972), 380 ~ 388; V. S. Linsky, *Vychisl. Mat.* **2** (1957), 90 ~ 119; D. E. Knuth, *METAFONT: The Program* (Reading, Mass.: Addison-wesley, 1986), §120 ~ §147。

29. e; 3; 4。

1.2.3 小节

1. $a_1 + a_2 + a_3$。

2. $\frac{1}{1}+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}+\frac{1}{9}+\frac{1}{11}$; $\frac{1}{9}+\frac{1}{3}+\frac{1}{1}+\frac{1}{3}+\frac{1}{9}$。

3. 违背了对于 $p(j)$ 的规则:首先, $n^2 = 3$ 没有 n 出现;其次, $n^2 = 4$ 出现有两个 n。[见等式(18)。]

4. $(a_{11}) + (a_{21} + a_{22}) + (a_{31} + a_{32} + a_{33}) = (a_{11} + a_{21} + a_{31}) + (a_{22} + a_{32}) + (a_{33})$。

5. 仅仅需要使用规则 $a\sum_{R(i)}x_i=\sum_{R(i)}(ax_i)$：

$$\Big(\sum_{R(i)}a_i\Big)\Big(\sum_{S(j)}b_j\Big)=\sum_{R(i)}a_i\Big(\sum_{S(j)}b_j\Big)=\sum_{R(i)}\Big(\sum_{S(j)}a_ib_j\Big)$$

7. 用等式(3)；交换两个极限，而且 a_0 与 a_c 之间的项必须从一个极限移到另一个。

8. 对所有 $i>0$，令 $a_{(i+1)i}=+1$ 及 $a_{i(i+1)}=-1$，且所有其它的 a_{ij} 为 0；令 $R(i)=S(i)=$ "$i\geqslant 0$"。左边为 -1，右边为 $+1$。

9,10. 否；假定 $n\geqslant 0$，应用规则(d)。（对于 $n=-1$ 结果是正确的，但推导不对。）

11. $(n+1)a$。

12. $\frac{7}{6}(1-1/7^{n+1})$。

13. $m(n-m+1)+\frac{1}{2}(n-m)(n-m+1)$；或者，$\frac{1}{2}(n(n+1)-m(m-1))$。

14. 如果 $m\leqslant n$ 且 $r\leqslant s$，则 $\frac{1}{4}(n(n+1)-m(m-1))(s(s+1)-r(r-1))$。

15,16. 关键步骤：

$$\sum_{0\leqslant j\leqslant n}jx^j=x\sum_{1\leqslant j\leqslant n}jx^{j-1}=x\sum_{0\leqslant j\leqslant n-1}(j+1)x^j=$$
$$x\sum_{0\leqslant j\leqslant n}jx^j-nx^{n+1}+x\sum_{0\leqslant j\leqslant n-1}x^j$$

17. S 中的元素数。

18. $S'(j)=$"$1\leqslant j<n$"。$R'(i,j)=$"n 是 i 的倍数而且 $i>j$"。

19. a_n-a_{m-1}。

20. $(b-1)\sum_{k=0}^{n}(n-k)b^k+n+1=\sum_{k=0}^{n}b^k$；这个公式是从(14)和习题 16 的结果得出的。

21. $\sum_{R(j)}a_j+\sum_{S(j)}a_j=\sum_ja_j[R(j)]+\sum_ja_j[S(j)]=\sum_ja_j([R(j)]+[S(j)])$；现在使用 $[R(j)]+[S(j)]=[R(j)$或$S(j)]+[R(j)$和$S(j)]$的事实。一般说来，方括号的记号给了我们"在行中"而不是"在行下"操作的能力。

22. 对于(5)和(7)，只要把 $\sum$ 换成 $\prod$ 即可。而且，我们还有 $\prod_{R(i)}b_ic_i=(\prod_{R(i)}b_i)(\prod_{R(i)}c_i)$ 且

$$\Big(\prod_{R(j)}a_j\Big)\Big(\prod_{S(j)}a_j\Big)=\Big(\prod_{R(j)\text{或}S(j)}a_j\Big)\Big(\prod_{R(j)\text{和}S(j)}a_j\Big)$$

23. $0+x=x$ 和 $1\cdot x=x$。这就使得许多运算和方程更简单，例如，规则(d)及其在上题中之类似者。

25. 第一步和最后一步是正确的。第二步，同时将 i 用于两种不同的目的。第三步，大概应该是 $\sum_{i=1}^{n}n$。

26. 在把此问题像在例 2 中那样变换之后的关键步骤：

$$\prod_{i=0}^{n}\Big(\prod_{j=0}^{n}a_ia_j\Big)=\prod_{i=0}^{n}\Big(a_i^{n+1}\prod_{j=0}^{n}a_j\Big)=$$
$$\Big(\prod_{i=0}^{n}a_i^{n+1}\Big)\Big(\prod_{i=0}^{n}\Big(\prod_{j=0}^{n}a_j\Big)\Big)=\Big(\prod_{i=0}^{n}a_i\Big)^{2n+2}$$

答案为 $(\prod_{i=0}^{n}a_i)^{n+2}$。

28. $(n+1)/2n$。

29. (a) $\sum_{0\leqslant k\leqslant j\leqslant i\leqslant n}a_ia_ja_k$。(b)令 $S_r=\sum_{i=0}^{n}a_i^r$。解：$\frac{1}{3}S_3+\frac{1}{2}S_1S_2+\frac{1}{6}S_1^3$。随着下标的个

数变得更大，这个问题的通解可在 1.2.9 小节中找到，即等式(38)。

30. 把左边写成$\sum_{1\leqslant j,k\leqslant n}a_jb_kx_jy_k$，并对右边作类似的事。（这个等式是习题 46 的 $m=2$ 的特殊情况。）

31. 置 $a_j=u_j, b_j=1, x_j=u_j$，以及 $y_j=1$，就可得到答案 $n\sum_{j=1}^n u_jv_j-(\sum_{j=1}^n u_j)(\sum_{j=1}^n v_j)$。

33. 对 n 使用归纳法就可以证明，但要把公式重新写成

$$\frac{1}{x_n-x_{n-1}}\left(\sum_{j=1}^n\frac{x_j^r(x_j-x_{n-1})}{\prod_{1\leqslant k\leqslant n,k\neq j}(x_j-x_k)}-\sum_{j=1}^n\frac{x_j^r(x_j-x_n)}{\prod_{1\leqslant k\leqslant n,k\neq j}(x_j-x_k)}\right)$$

这些和式中的每个都具有原有和式的形式，但对 $n-1$ 个元素除外，当 $0\leqslant 1\leqslant n-1$ 时，通过归纳法其值美妙地出现。当 $r=n$ 时，请考虑恒等式

$$0=\sum_{j=1}^n\frac{\prod_{k=1}^n(x_j-x_k)}{\prod_{1\leqslant k\leqslant n,k\neq j}(x_j-x_k)}=\sum_{j=1}^n\frac{x_j^n-(x_1+\cdots+x_n)x_j^{n-1}+P(x_j)}{\prod_{1\leqslant k\leqslant n,k\neq j}(x_j-x_k)}$$

其中 $P(x_j)$是 $n-2$ 次多项式；根据 $r=0,1,\cdots,n-1$ 时的解，就可以获得所需要的解。

注：Matrix 博士的这一发现被 L. Euler 抢先一步。欧拉在 1762 年 11 月 9 日就为此而写信给 Christian Goldbach。请参见他的*Institutionum Calculi Integralis* **2** (1769)，§ 1169；以及 E. Waring,*Phil. Trans.* **69** (1779)，64 ~ 67。下面的另一种证明方法使用了复变函数理论，不是太初等但更优美：根据留数定理，给定和式的值为

$$\frac{1}{2\pi i}\int_{|z|=R}\frac{z^r dz}{(z-x_1)\cdots(z-x_n)}$$

其中 $R>|x_1|,\cdots,|x_n|$。积分式的 Laurent 展开在$|z|=R$ 上均匀收敛，它就是

$$z^{r-n}\left(\frac{1}{1-x_1/z}\right)\cdots\left(\frac{1}{1-x_n/z}\right)=$$

$$z^{r-n}+(x_1+\cdots+x_n)z^{r-n-1}+(x_1^2+x_1x_2+\cdots)z^{r-n-2}+\cdots$$

逐项进行积分，除了 z^{-1}的系数之外，一切全都为 0。这个方法，给了我们对于任意整数 $r\geqslant 0$ 的一般公式

$$\sum_{\substack{j_1+\cdots+j_n=r-n+1\\ j_1,\cdots,j_n\geqslant 0}}x_1^{j_1}\cdots x_n^{j_n}$$

[J. J. Sylvester,*Quart. J. Math.* **1** (1857)，141 ~ 152。]

34. 如果读者已经非常认真地来解决这个问题，但却得不到答案，大概这个题目的目的就已经达到了。把分子看做 x 的多项式而不是看做k 的多项式，这种倾向几乎占绝对优势。毫无疑问地，更易于证明颇为一般的结果

$$\sum_{k=1}^n\frac{\prod_{1\leqslant r\leqslant n-1}(y_k-z_r)}{\prod_{1\leqslant r\leqslant n,r\neq k}(y_k-y_r)}=1$$

这是 $2n-1$ 个变量的恒等式。

35. 如果 $R(j)$根本不成立，则这个值应是 $-\infty$。所述的规则 a)的类似性，是以恒等式 $a+\max(b,c)=\max(a+b,a+c)$为基础的。类似地，如果所有的 a_i, b_j 都非负，则我们有

$$\sup_{R(i)} a_i \sup_{S(j)} b_j = \sup_{R(i)} \sup_{S(j)} a_i b_j$$

规则 b),c)不变;对于规则 d),我们得到更简单的形式

$$\sup(\sup_{R(j)} a_j, \sup_{S(j)} a_j) = \sup_{R(j) \text{ or } S(j)} a_j$$

36. 从第 2,…,n 列减去第 1 列。把第 2,…,n 行加到第 1 行。结果是三角行列式。

37. 从第 2,…,n 列减去第 1 列。然后对于 $k=n,n-1,\cdots,2$(按此顺序),从行 k 减去 x_1 乘行 $k-1$。我们现在从第 1 列提取因子 x_1,而且从列 $k=2,\cdots,n$ 提取因子 x_k-x_1,得到 $x_1(x_2-x_1)\cdots(x_n-x_1)$乘 $n-1$ 阶的范德蒙德行列式,这个过程通过归纳法而继续下去。

另一种证明方法,利用“更高等的”数学:该行列式是总次数为 $1+2+\cdots+n$ 的关于变量 $x_1,\cdots,x_n$ 的多项式。如果 $x_j=0$ 或者如果 $x_i=x_j(i<j)$,则它就为 0。而且 $x_1^1x_2^2\cdots x_n^n$ 的系数为 $+1$。这些事实就表征了它的值。一般地,如果矩阵的两行由于 $x_i=x_j$ 而变为相等时,则它们的差总是可为 x_i-x_j 整除,而这个发现通常加快行列式的计算。

38. 从第 2,…,n 列减去第 1 列,并从诸行和诸列提取因子$(x_1+y_1)^{-1}\cdots(x_n+y_1)^{-1}(y_1-y_2)\cdots(y_1-y_n)$。现在从第 2,…,$n$ 行减去第 1 行,并提出因子$(x_1-x_2)\cdots(x_1-x_n)(x_1+y_2)^{-1}\cdots(x_1+y_n)^{-1}$;从而得到 $n-1$ 阶的柯西行列式。

39. 命 I 为单位矩阵(δ_{ij}),J 是元素全为 1 的矩阵。由于 $J^2=nJ$,我们有

$$(xI+yJ)((x+ny)I-yJ)=x(x+ny)I$$

40. $\sum_{t=1}^{n} b_{it}x_j^t = x_j \prod_{\substack{1\le k\le n\\ k\ne i}}(x_k-x_j)\Big/ x_i \prod_{\substack{1\le k\le n\\ k\ne i}}(x_k-x_i) = \delta_{ij}$。

41. 由逆矩阵与它的余子式的关系,可立即得出。这里给出的直接证明可能也是有趣的:当 $x=0$ 时我们有

$$\sum_{t=1}^{n}\frac{1}{x_i+y_t}b_{tj} = \sum_{t=1}^{n}\frac{\prod_{k\ne t}(x_j+y_k-x)\prod_{k\ne i}(x_k+y_t)}{\prod_{k\ne j}(x_j-x_k)\prod_{k\ne t}(y_t-y_k)}$$

这是 x 的至多$n-1$ 次的多项式。如果我们置 $x=x_j+y_s$,$1\le s\le n$,则这些项都为 0,但 $s=t$ 时除外,所以这个多项式的值为

$$\prod_{k\ne i}(-x_k-y_s)\Big/\prod_{k\ne j}(x_j-x_k) = \prod_{k\ne i}(x_j-x_k-x)\Big/\prod_{k\ne j}(x_j-x_k)$$

由于这些次数至多为 $n-1$ 的多项式,在 n 个不同的 x 上一致,所以它们对于 $x=0$ 也一致,因此

$$\sum_{t=1}^{n}\frac{1}{x_i+y_t}b_{tj} = \prod_{k\ne i}(x_j-x_k) = \delta_{ij}$$

42. $n/(x+ny)$。

43. $1-\prod_{k=1}^{n}(1-1/x_k)$。如果有任何 $x_i=1$,这是容易验证的,因为任何有一行或一列全为 1 的矩阵之逆,必然有其和为 1 的一些元素。如果没有 x_i 等于 1,则像在习题 44 中那样把第 i 行的元素加起来,并得到$\prod_{k\ne i}(x_k-1)/x_i\prod_{k\ne i}(x_k-x_i)$。我们现在使用习题 33 能对 i 求和,且 $r=0$(以(x_i-1)乘分子和分母)。

44. 在应用习题 33 之后,我们求出

$$c_j = \sum_{i=1}^{n} b_{ij} = \prod_{k=1}^{n}(x_j+y_k)\Big/\prod_{\substack{1\le k\le n\\ k\ne j}}(x_j-x_k)$$

而且

$$\sum_{j=1}^{n} c_j = \sum_{j=1}^{n} \frac{(x_j^n + (y_1 + \cdots + y_n)x_j^{n-1} + \cdots)}{\prod_{1\le k\le n,\,k\ne j}(x_j - x_k)} =$$

$$(x_1 + x_2 + \cdots + x_n) + (y_1 + y_2 + \cdots + y_n)$$

45. 命 $x_i = i$, $y_j = j - 1$。由习题 44,逆矩阵的元素之和为 $(1 + 2 + \cdots + n) + ((n-1) + (n-2) + \cdots + 0) = n^2$。由习题 38,逆矩阵的元素是

$$b_{ij} = \frac{(-1)^{i+j}(i+n-1)!(j+n-1)!}{(i+j-1)(i-1)!^2(j-1)!^2(n-i)!(n-j)!}$$

这个量可以写成与二项式系数有关的几种形式,例如

$$\frac{(-1)^{i+j}ij}{i+j-1}\binom{-i}{n}\binom{n}{i}\binom{-j}{n}\binom{n}{j} = (-1)^{i+j}j\binom{i+j-2}{i-1}\binom{i+n-1}{i-1}\binom{j+n-1}{n-i}\binom{n}{j}$$

从这后边的公式,我们看到,b_{ij}不仅是一个整数,而且它还可被 $i, j, n, i+j-1, i+n-1, j+n-1, n-i+1$ 以及 $n-j+1$ 所整除。也许对于 b_{ij}的最好的公式是

$$(i+j-1)\binom{i+j-2}{i-1}^2\binom{-(i+j)}{n-i}\binom{-(i+j)}{n-j}$$

如果没有认识到希尔伯特矩阵乃是柯西矩阵的特殊情况,那么,解决这个问题将是极为困难的;更一般的问题要比它的特殊情况容易解决得多!把问题推广成为它的"归纳法闭包",即最小的推广,使得在企图用数学归纳法做出的证明中,所产生的全部子问题都属于这同类中,这样做往往是明智的。在这种情况下,我们看到,柯西矩阵的余子式是柯西矩阵,但希尔伯特矩阵的余子式则不是希尔伯特矩阵。[关于进一步的介绍,见 J. Todd, *J. Res. Nat. Bur. Stand.* **65** (1961), 19 ~ 22。]

46. 对于任何整数 $k_1, k_2, \cdots, k_m$,令 $\epsilon(k_1, \cdots, k_m) = \text{sign}(\prod_{1\le i<j\le m}(k_j - k_i))$,其中 $\text{sign}\, x = [x>0] - [x<0]$。如果$(l_1, \cdots, l_m)$等于$(k_1, \cdots, k_m)$,除了 k_i 和 k_j 已被交换这一事实外,我们有 $\epsilon(l_1, \cdots, l_m) = -\epsilon(k_1, \cdots, k_m)$。因此如果 $j_1 \le \cdots \le j_m$ 是重新安排成非递减顺序的数 $k_1, \cdots, k_m$,那我们有等式 $\det(B_{k_1\cdots k_m}) = \epsilon(k_1, \cdots, k_m)\det(B_{j_1\cdots j_m})$。现在由行列式的定义,

$$\begin{aligned}
\det(AB) = &\sum_{1\le l_1,\cdots,l_m\le m} \epsilon(l_1, \cdots, l_m)\Big(\sum_{k=1}^{n} a_{1k}b_{kl_1}\Big)\cdots\Big(\sum_{k=1}^{n} a_{mk}b_{kl_m}\Big) = \\
&\sum_{1\le k_1,\cdots,k_m\le n} a_{1k_1}\cdots a_{mk_m} \sum_{1\le l_1,\cdots,l_m\le m} \epsilon(l_1, \cdots, l_m)b_{k_1l_1}\cdots b_{k_ml_m} = \\
&\sum_{1\le k_1,\cdots,k_m\le n} a_{1k_1}\cdots a_{mk_m}\det(B_{k_1\cdots k_m}) = \\
&\sum_{1\le k_1,\cdots,k_m\le n} \epsilon(k_1, \cdots, k_m)a_{1k_1}\cdots a_{mk_m}\det(B_{j_1\cdots j_m}) = \\
&\sum_{1\le j_1,\cdots,j_m\le n} \det(A_{j_1\cdots j_m})\det(B_{j_1\cdots j_m})
\end{aligned}$$

最后,如果两个 j 相等,则 $\det(A_{j_1\cdots j_m}) = 0$。[*J. de l'École Polytechnique* **9** (1813), 280 ~ 354; **10** (1815), 29 ~ 112。Binet 和 Cauchy 在 1812 年在同一天里提交了他们的文章。]

47. 令 $a_{ij}=(\prod_{k=1}^{j-1}(x_i+p_k))(\prod_{k=j+1}^{n}(x_i+q_k))$。对于 $k=n,n-1,\cdots,j+1$(按此顺序)和 $j=1,2,\cdots,n-1$(按此顺序),从列 k 中减去列 $k-1$ 并提取因子 $p_{k-j}-q_k$。这得出 $\prod_{1\leqslant i<j\leqslant n}(p_i-q_j)$ 乘以 $\det(b_{ij})$,其中 $b_{ij}=\prod_{k=j+1}^{n}(x_i+q_k)$。现在对于 $k=1,\cdots,n-j$ 和对于 $j=1,\cdots,n-1$,从列 k 中减去 q_{k+j} 乘列 $k+1$;这得出 $\det(c_{ij})$,其中 $c_{ij}=x_i^{n-j}$ 实质上定义了一个范德蒙德矩阵。现在我们可以像在习题 37 中那样进行,对于行而不是对列进行运算,得到

$$\det(a_{ij})=\prod_{1\leqslant i<j\leqslant n}(x_i-x_j)(p_i-q_j)$$

当对于 $1\leqslant j\leqslant n$, $p_j=q_j=y_j$ 时,本题中的矩阵是柯西矩阵且第 i 行被乘以 $\prod_{j=1}^{n}(x_i+y_j)$。因此通过加上 $n-1$ 个独立的参数本结果推广了习题 38。[*Manuscripta Math*. **69** (1990), 177 ~ 178。]

1.2.4 小节

1. 1, −2, −1, 0, 5。

2. $\lfloor x\rfloor$。

3. 由定义,$\lfloor x\rfloor$是小于或等于 x 的最大整数;因此,$\lfloor x\rfloor$为一整数,$\lfloor x\rfloor\leqslant x$,而且$\lfloor x\rfloor+1>x$。这后一性质,加上这样一个事实,即当 m 和 n 为整数时,$m<n$ 当且仅当 $m\leqslant n-1$,就容易导出命题 a)和 b)的证明。类似的论证可证明 c)和 d)。最后,e)和 f)只不过是本题的前几部分之组合。

4. $x-1<\lfloor x\rfloor\leqslant x$;所以 $-x+1>-\lfloor x\rfloor\geqslant -x$;从而得出结果。

5. $\left\lfloor x+\frac{1}{2}\right\rfloor$。($-x$ 舍入)的值与 $-$(x 舍入)是相同的,除非当 $x \bmod 1=\frac{1}{2}$ 时。后一情况负的值舍入时向 0 方向靠近,而正的值舍入时则远离 0。

6. (a)为真:$\lfloor\sqrt{x}\rfloor=n\Leftrightarrow n^2\leqslant x<(n+1)^2\Leftrightarrow n^2\leqslant\lfloor x\rfloor<(n+1)^2\Leftrightarrow\lfloor\sqrt{\lfloor x\rfloor}\rfloor=n$。类似地,(b)为真。但(c)为假,例如当 x 为 1.1 时。

7. $\lfloor x+y\rfloor=\lfloor\lfloor x\rfloor+x \bmod 1+\lfloor y\rfloor+y \bmod 1\rfloor=\lfloor x\rfloor+\lfloor y\rfloor+\lfloor x \bmod 1+y \bmod 1\rfloor$,对于顶限函数来说,不等式应是≥,而当且仅当 x 或 y 为整数,或者 $x \bmod 1+y \bmod 1>1$ 时等式成立。

8. 1,2,5, −100。

9. −1,0, −2。

10. 0.1,0.01, −0.09。

11. $x=y$。

12. 所有的整数。

13. +1, −1。

14. 8。

15. 以 z 来乘等式(1)的两边;如果 $y=0$,这一结果也容易验证。

17. 作为一个例子,考虑定律 A 的乘法部分:对于某整数 q 和 r 我们有 $a=b+qm$, $x=y+rm$;所以 $ax=by+(br+yq+qrm)m$。

18. 对于某个整数 k,我们有 $a-b=kr$,而且还有 $kr\equiv 0$(modulo s)。因此,由定律 B, $k\equiv 0$ (modulo s),所以对于某个整数 q, $a-b=qsr$。

20. 以 a' 来乘同余式的两边。

21. 按以前证明的习题,至少有一种这样的表示。如果有两种表示,$n=p_1\cdots p_k=q_1\cdots q_m$,则我们有 $q_1\cdots q_m\equiv 0$(modulo p_1);所以如果所有的 q 都不等于 p_1,则由定律 B 我们就可以全部消去诸 q 并得到 $1\equiv 0$(modulo p_1)。后者是不可能的,因为 p_1 不等于 1。所以某个 q_j 等于 p_1,而且 $n/p_1=p_2\cdots p_k=q_1\cdots q_{j-1}q_{j+1}\cdots q_m$。每个 n 都是素数,这时结果显然为真,或者通过归纳法,n/p_1

的两个因子分解相同。

22. 令 $m=ax$,其中 $a>0$ 和 $x>0$。则 $ax\equiv 0$ 但 $x\not\equiv 0 \pmod{m}$。

24. 对于加法和减法来说,定律 A 总是对的;定律 C 总是对的。

26. 如果 b 不是 p 的倍数,则 b^2-1 是 p 的倍数,所以它的因子之一也必定是 p 的倍数。

27. 一个数与 p^e 互素,当且仅当它不是 p 的倍数。所以我们统计不是 p 的倍数的那些数,并得到 $\varphi(p^e)=p^e-p^{e-1}$。

28. 如果 a 和 b 与 m 互素,则 $(ab \bmod m)$ 也与 m 互素,因为整除后者和 m 的任何素数必然也整除 a 或 b。现在简单地令 $x_1,\cdots,x_{\varphi(m)}$ 是与 m 互素的数,并观察 $ax_1 \bmod m,\cdots,ax_{\varphi(m)} \bmod m$ 是在某一顺序下相同的数,等等。

29. 我们证明(b):如果 $r\perp s$ 而且如果 k^2 整除 rs,那么对于某个素数 p,p^2 整除 rs,所以 p 整除 r(比如说)但不能整除 s;所以 p^2 整除 r。我们看到,$f(rs)=0$,当且仅当 $f(r)=0$ 或 $f(s)=0$。

30. 设 $r\perp s$。一种想法是证明,$\varphi(rs)$ 个与 rs 互素的数,恰恰就是 $\varphi(r)\varphi(s)$ 个不同的数 $(sx_i+ry_j) \bmod (rs)$,其中 $x_1,\cdots,x_{\varphi(r)}$ 和 $y_1,\cdots,y_{\varphi(s)}$ 是对于 r 和 s 的对应的值。

由于 φ 是乘性的,$\varphi(10^6)=\varphi(2^6)\varphi(5^6)=(2^6-2^5)(5^6-6^5)=400000$。而且一般说来,当 $n=p_1^{e_1}\cdots p_r^{e_r}$ 时,我们有 $\varphi(n)=(p_1^{e_1}-p_1^{e_1-1})\cdots(p_r^{e_r}-p_r^{e_r-1})=n\prod_{p\backslash n,p\text{为素数}}(1-1/p)$。(另一种证明请见习题 1.3.3-27。)

31. 利用 rs 的因子可以惟一地写成 cd 的形式这一事实,其中 c 整除 r 且 d 整除 s。类似地,如果 $f(n)\geqslant 0$,可以证明函数 $\max_{d\backslash n}f(d)$ 是乘性的(见习题 1.2.3-35)。

33. 要么 $n+m$,要么 $n+m+1$ 为偶数,所以左边方括号内的数量之一是整数;从而习题 7 的等式成立,而且我们得到(a) n;(b) $n+1$。

34. b 必须是大于等于 2 的整数。(置 $x=b$。)充分性的证明就像在习题 6 中那样。同样的条件对于 $\lceil\log_b x\rceil=\lceil\log_b\lceil x\rceil\rceil$ 是充分必要的。

注:R. J. McEliece 给出了下列推广:设 f 是定义在区间 A 上的连续和严格地递增的函数,并且假定每当 x 在 A 中时 $\lfloor x\rfloor$ 和 $\lceil x\rceil$ 也都在 A 中。于是对于 A 中所有 x,关系 $\lfloor f(x)\rfloor=\lfloor f(\lfloor x\rfloor)\rfloor$ 成立,当且仅当对于 A 中所有 x,关系 $\lceil f(x)\rceil=\lceil f(\lceil x\rceil)\rceil$ 成立,当且仅当对于 A 中所有 x 下列条件满足:"$f(x)$ 是整数意味着 x 是整数。"这个条件显然是必要的,因为如果 $f(x)$ 是一个整数,而且它等于 $\lfloor f(\lfloor x\rfloor)\rfloor$ 或 $\lceil f(\lceil x\rceil)\rceil$,则 x 必然等于 $\lfloor x\rfloor$ 或 $\lceil x\rceil$。反之,比如说 $\lfloor f(\lfloor x\rfloor)\rfloor<\lfloor f(x)\rfloor$,那么根据连续性,就存在满足 $\lfloor x\rfloor<y\leqslant x$ 的某个 y,使 $f(y)$ 为整数;但是 y 不可能是整数。

35. $\dfrac{x+m}{n}-1=\dfrac{x+m}{n}-\dfrac{1}{n}-\dfrac{n-1}{n}<\dfrac{\lfloor x\rfloor+m}{n}-\dfrac{n-1}{n}\leqslant\left\lfloor\dfrac{\lfloor x\rfloor+m}{n}\right\rfloor\leqslant\dfrac{x+m}{n}$;应用习题 3。用习题 4 给出类似的关于顶限函数之结果。两个恒等式遂作为习题 34 答案中的 McEliece 定理的特殊情况而得出。

36. 首先假定 $n=2t$。于是

$$\sum_{k=1}^{n}\left\lfloor\frac{k}{2}\right\rfloor=\sum_{k=1}^{n}\left\lfloor\frac{n+1-k}{2}\right\rfloor$$

因此根据习题 33 有

$$\sum_{k=1}^{n}\left\lfloor\frac{k}{2}\right\rfloor=\frac{1}{2}\sum_{k=1}^{n}\left(\left\lfloor\frac{k}{2}\right\rfloor+\left\lfloor\frac{n+1-k}{2}\right\rfloor\right)=\frac{1}{2}\sum_{k=1}^{n}\left\lfloor\frac{2t+1}{2}\right\rfloor=t^2=\frac{n^2}{4}$$

而且,如果 $n=2t+1$,我们有 $t^2+\lfloor n/2\rfloor=t^2+t=n^2/4-1/4$。对于第二个和,类似地得到 $\lceil n(n+2)/4\rceil$。

37. $\sum_{0\le k<n}\frac{mk+x}{n}=\frac{m(n-1)}{2}+x$。令$\{y\}$表示 $y \bmod 1$;必须减去

$$S=\sum_{0\le k<n}\left\{\frac{mk+x}{n}\right\}$$

这个量 S 由相同和式的 d 个副本所组成,因为如果 $t=n/d$,得到

$$\left\{\frac{m+k+x}{n}\right\}=\left\{\frac{m(k+t)+x}{n}\right\}$$

令 $u=m/d$;则

$$\sum_{0\le k<t}\left\{\frac{mk+x}{n}\right\}=\sum_{0\le k<t}\left\{\frac{x}{n}+\frac{uk}{t}\right\}$$

而且因为 $t\perp u$,因此这个式可重新排列为等于

$$\left\{\frac{x \bmod d}{n}\right\}+\left\{\frac{x \bmod d}{n}+\frac{1}{t}\right\}+\cdots+\left\{\frac{x \bmod d}{n}+\frac{t-1}{t}\right\}$$

最后,由于$(x \bmod d)/n<1/t$,可消去和式中的大括弧,因而我们有

$$S=d\left(\frac{t(x \bmod d)}{n}+\frac{t-1}{2}\right)$$

应用习题 4,得到类似的恒等式

$$\sum_{0\le k<n}\left\lceil\frac{mk+x}{n}\right\rceil=\frac{(m+1)(n-1)}{2}-\frac{d-1}{2}+d\lceil x/d\rceil$$

这个公式将对 m 和 n 变成对称的,如果把它扩充到范围 $0<k\le n$。(对称性可以以下述方法来阐明:即把被加项作为 k 的函数画出图形,然后关于直线 $y=x$ 进行反射。)

38. 当 x 增加 1 时,两边都增加$\lceil y\rceil$,所以我们可以假定 $0\le x<1$。然后当 $x=0$ 时,两边都为零,而且当对于 $y>k>0$,x 增加超过 $1-k/y$ 的值时,两边都增加 1。[*Crell* **136** (1909),42;$y=n$ 的情况是由 C. Hermite 给出的,*Acta Math*. **5** (1884),315。]

39. f)的证明:考虑更一般的恒等式$\prod_{0\le k<n}2\sin\pi(x+k/n)=2\sin\pi nx$,可以证明如下:由于 $2\sin\theta=(e^{i\theta}-e^{-i\theta})/i=(1-e^{-2i\theta})e^{i\theta-i\pi/2}$,这恒等式是以下两个公式的推论:

$$\prod_{0\le k<n}(1-e^{-2\pi(x+ik/n)})=1-e^{-2\pi nx}\ \text{和}\ \prod_{0\le k<n}e^{\pi(x-(1/2)+(k/n))}=e^{\pi(nx-1/2)}$$

后者为真,因为函数 $x-\frac{1}{2}$ 是重叠的;前者为真,因为在多项式的因子分解 $z^n-\alpha^n=(z-\alpha)(z-\omega\alpha)\cdots(z-\omega^{n-1}\alpha)$之中,我们可以置 $z=1$,其中 $\omega=e^{-2\pi i/n}$。

40.(由 N. G. de Bruijn 说明)如果 f 是重叠的,则 $f(nx+1)-f(nx)=f(x+1)-f(x)$对于所有 $n>0$ 都成立。因此,如果 f 是连续的,则 $f(x+1)-f(x)=c$ 对于所有的 x 成立,而且 $g(x)=f(x)-c\lfloor x\rfloor$是重叠的和周期的。现在

$$\int_0^1 e^{2\pi inx}g(x)\,dx=\frac{1}{n}\int_0^1 e^{2\pi iy}g(y)\,dy$$

展开成傅里叶级数可以看出,对于 $0<x<1$,$g(x)=\left(x-\frac{1}{2}\right)a$。由此推出 $f(x)=\left(x-\frac{1}{2}\right)a$。一般地,这个论证说明了,任何重叠的局部黎曼可积函数几乎处处都有形式

$$\left(x-\frac{1}{2}\right)a+b\ \max(\lfloor x\rfloor,0)+c\ \min(\lfloor x\rfloor,0)$$

关于进一步的结果，请见 L. J. Mordell, *J. London Math. Soc.* **33** (1958), 371 ~ 375; M. F. Yoder, *Æquationes Mathematicæ* **13** (1975), 251 ~ 261。

41. 当$\frac{1}{2}k(k-1)<n\leqslant\frac{1}{2}k(k+1)$时，我们要求 $a_n=k$。因为 n 是整数，这等价于

$$\frac{k(k-1)}{2}+\frac{1}{8}<n<\frac{k(k+1)}{2}+\frac{1}{8}$$

即 $k-\frac{1}{2}<\sqrt{2n}<k+\frac{1}{2}$。因此 $a_n=\left\lfloor\sqrt{2n}+\frac{1}{2}\right\rfloor$，即最接近于$\sqrt{2n}$的整数。另外的正确答案为$\lceil(\sqrt{2n})-\frac{1}{2}\rceil$，$\lceil(\sqrt{8n+1}-1)/2\rceil$和$\lfloor(\sqrt{8n-7}+1)/2\rfloor$，等等。

42. (a) 参照习题 1.2.7-10。(b) 给出的和为 $n\lfloor\log_b n\rfloor-S$，其中

$$S=\sum_{\substack{1\leqslant k<n\\ k+1\text{为}b\text{的幂}}}k=\sum_{1\leqslant t\leqslant\log_b n}(b^t-1)=(b^{\lfloor\log_b n\rfloor+1}-b)/(b-1)-\lfloor\log_b n\rfloor$$

43. $\lfloor\sqrt{n}\rfloor\left(n-\frac{1}{6}(2\lfloor\sqrt{n}\rfloor+5)(\lfloor\sqrt{n}\rfloor-1)\right)$。

44. 当 n 为负时，这个和为 $n+1$。

45. $\lfloor mj/n\rfloor=r$ 当且仅当 $\left\lceil\frac{rn}{m}\right\rceil\leqslant j<\left\lceil\frac{(r+1)n}{m}\right\rceil$，而且我们发现，给定的和因此是

$$\sum_{0\leqslant r<m}f(r)\left(\left\lceil\frac{(r+1)n}{m}\right\rceil-\left\lceil\frac{rn}{m}\right\rceil\right)$$

通过重新排列这后边的和，把具有特定的$\lceil rn/m\rceil$之值的那些项归在一起，而得到所述结果。第二个公式通过代换

$$f(x)=\binom{x+1}{k}$$

立即得到。

46. $\sum_{0\leqslant j<\alpha n}f(\lfloor mj/n\rfloor)=\sum_{0\leqslant r<\alpha m}\lceil rn/m\rceil(f(r-1)-f(r))+\lceil\alpha n\rceil f\lceil\alpha m\rceil-1)$。

47. a) 数 $2,4,\cdots,p-1$ 是模 p 偶留数；因为 $2kq=p\lfloor 2kq/p\rfloor+(2kq)\bmod p$，数$(-1)^{\lfloor 2kq/p\rfloor}((2kq)\bmod p)$将是偶留数或偶留数减 p，而且每一个偶留数显然地恰好出现一次。因此 $(-1)^\sigma q^{(p-1)/2}2\cdot4\cdots(p-1)\equiv2\cdot4\cdots(p-1)$。

b) 令 $q=2$。如果 $p=4n+1$，则 $\sigma=n$；如果 $p=4n+3$，$\sigma=n+1$。因此分别根据 $p\bmod 8=(1,3,5,7)$，$\left(\frac{2}{p}\right)=(1,-1,-1,1)$。

c) 对于 $k<p/4$，我们有

$$\lfloor(p-1-2k)q/p\rfloor=q-\lceil(2k+1)q/p\rceil=q-1-\lfloor(2k+1)q/p\rfloor\equiv\lfloor(2k+1)q/p\rfloor(\text{modulo }2)$$

因此我们可以以$\lfloor q/p\rfloor$，$\lfloor 3q/p\rfloor$等等代替最后的项$\lfloor(p-1)q/p\rfloor$，$\lfloor(p-3)q/p\rfloor$，…。
d) $\sum_{0\leqslant k<p/2}\lfloor kq/p\rfloor+\sum_{0\leqslant r<q/2}\lceil rp/q\rceil=\lceil p/2\rceil(\lceil q/2\rceil-1)=(p+1)(q-1)/4$。同时 $\sum_{0\leqslant r<q/2}\lceil rp/q\rceil=\sum_{0\leqslant r<q/2}\lfloor rp/q\rfloor+(q-1)/2$。这个证明的思想可追溯到 G. Eisenstein, *Crelle*

28 (1844),246~248。在同一卷中,Eisenstein 还给出这个和其它互反定律的若干其它的证明。

48.(a) 当 $n<0$ 时,显然地这不总是成立的;当 $n>0$ 时这容易验证。(b) $\lfloor(n+2-\lfloor n/25\rfloor)/3\rfloor=\lceil(n-\lfloor n/25\rfloor)/3\rceil=\lceil(n+\lceil -n/25\rceil)/3\rceil=\lceil\lceil 24n/25\rceil/3\rceil=\lceil 8n/25\rceil=\lfloor(8n+24)/25\rfloor$。倒数第二个等式可由习题 35 证明。

49. 由于 $f(0)=f(f(0))=f(f(0)+0)=f(0)+f(0)$,对于所有整数 n 我们有 $f(n)=n$。如果 $f\left(\frac{1}{2}\right)=k\leq 0$,我们有 $k=f\left(\frac{1}{1-2k}f\left(\frac{1}{2}-k\right)\right)=f\left(\frac{1}{1-2k}\left(f\left(\frac{1}{2}\right)-k\right)\right)=f(0)=0$。而且如果 $f\left(\frac{1}{n-1}\right)=0$,我们有 $f\left(\frac{1}{n}\right)=f\left(\frac{1}{n}f\left(1+\frac{1}{n-1}\right)\right)=f\left(\frac{1}{n-1}\right)=0$; 其次 $1\leq m<n$ 意味着对于 $a=\lceil n/m\rceil$,由对 m 的归纳法,$f\left(\frac{m}{n}\right)=f\left(\frac{1}{a}f\left(\frac{am}{n}\right)\right)=f\left(\frac{1}{a}\right)=0$。因此 $f\left(\frac{1}{2}\right)\leq 0$ 意味着对于所有有理数 x,$f(x)=\lfloor x\rfloor$。另一方面,如果 $f\left(\frac{1}{2}\right)>0$,函数 $g(x)=-f(-x)$满足(i)和(ii)且有 $g\left(\frac{1}{2}\right)=1-f\left(\frac{1}{2}\right)\leq 0$;因此对于所有有理数 x,$f(x)=-g(-x)=-\lfloor -x\rfloor=\lceil x\rceil$。[P. Eisele 和 K. P. Hadeler,*AMM* **97** (1990), 475~477。]

但是,不能由此得出,对于所有 x 的实数值,有 $f(x)=\lfloor x\rfloor$或$\lceil x\rceil$。如果,比如说,$h(x)$是对于所有实的 x 和 y 有 $h(1)=1$ 和 $h(x+y)=h(x)+h(y)$的任何函数,则函数 $f(x)=\lfloor h(x)\rfloor$满足(i)和(ii);但当 $0<x<1$ 时 $h(x)$可能是无界的和高度无规律的[G. Hamel,*Math. Annalen* **60** (1905),459~462。]

1.2.5 小节

1. 52!。要是好奇,这个数算出是 806 58175 17094 38785 71660 63685 64037 66975 28950 54408 83277 82400 00000 00000。

2. $p_{nk}=p_{n(k-1)}(n-k+1)$。在放置了前 $n-1$ 个对象之后,对于最后一个对象仅有一种可能性。

3. 53124,35124,31524,31254,31245;42351,41352,41253,31254,31245。

4. 有 2568 个数字。第一个数字是 4(因为 $\log_{10}4=2\log_{10}2\approx .602$)。最低有效位数字为 0,而且事实上由等式(8),低位的 249 个数字全都是零! H. S. Uhler 使用一台台式计算器并且依靠在若干年期间的极度的耐心,算出了 1000! 的精确值,这个值发表于*Scripta Mathematica* **21** (1955), 266~267。此数以 402 38726 00770…开始。(在计算中的最后一步,即把两个数 750! 和 $\prod_{k=751}^{1000}k$ 相乘,是由 John W. Wrench, Jr. 在 UNIVAC 计算机上,以"非同寻常的 2 ½分"的时间实现的。当然,今天,台式机在不到一秒的时间里就可以很容易地计算出 1000! 来,因此我们可以确信 Uhler 的值 100% 地正确。)

5. $(39902)(97/96)\approx 416+39902=40318$。

6. $2^{18}\cdot 3^8\cdot 5^4\cdot 7^2\cdot 11\cdot 13\cdot 17\cdot 19$。

8. 它是 $\lim_{m\to\infty}m^n m!/((n+m)!/n!)=n!\lim_{m\to\infty}m^n/((m+1)\cdots(m+n))=n!$,因为 $m/(m+k)\to 1$。

9. $\sqrt{\pi}$和 $-2\sqrt{\pi}$。(使用了习题 10。)

10. 对,除了当 x 为 0 或一负整数时外。因为我们有

$$\Gamma(x+1)=x\lim_{m\to\infty}\frac{m^x m!}{x(x+1)\cdots(x+m)}\left(\frac{m}{x+m+1}\right)$$

11,12. $\mu = (a_k p^{k-1} + \cdots + a_1) + (a_k p^{k-2} + \cdots + a_2) + \cdots + a_k =$
$a_k(p^{k-1} + \cdots + p + 1) + \cdots + a_1 = (a_k(p^k - 1) + \cdots + a_0(p^0 - 1))/(p-1) =$
$(n - a_k - \cdots - a_1 - a_0)/(p-1)$

13. 对于每个 $n, 1 \leqslant n < p$,如同在习题 1.2.4-19 中那样确定 n'。由定律 1.2.4B,我们恰好有一个这样的 n';而且 $(n')' = n$。因此我们能把诸数两两配对成两个一组,假定 $n' \neq n$。如果 $n' = n$,则我们有 $n^2 \equiv 1(\text{modulo } p)$;因此,如同在习题 1.2.4-26 中那样,$n = 1$ 或 $n = p - 1$。所以 $(p-1)! \equiv 1 \cdot 1 \cdot \cdots \cdot (-1)$,因为 1 和 $p-1$ 是仅有的未配对的元素。

14. 不为 p 之倍数的数 $\{1,2,\cdots,n\}$ 中,有 $\lfloor n/p \rfloor$ 个由 $p-1$ 个连续的元素组成的完备集合,由 Wilson 定理,每个都有模 p 同余于 -1 的乘积。而且还剩下 a_0,这部分模 p 同余于 $a_0!$;所以,不为 p 之倍数的那些因子的贡献是 $(-1)^{\lfloor n/p \rfloor} a_0!$。而来自是 p 的倍数 的那些因子的贡献等同于 $\lfloor n/p \rfloor!$ 中的贡献;因此可以重复这个论证以获得所求的公式。

15. $(n!)^3$。有 $n!$ 项。每一项均有来自每行每列的各一个元素,所以它有值 $(n!)^2$。

16. 这些项不趋于 0,因为系数趋于 $1/e$。

17. 通过等式(15)把伽马函数表达成极限。

18. $\displaystyle\prod_{n\geqslant 1} \frac{n}{n-\frac{1}{2}} \frac{n}{n+\frac{1}{2}} = \frac{\Gamma\left(\frac{1}{2}\right)\Gamma\left(\frac{3}{2}\right)}{\Gamma(1)\Gamma(1)} = 2\Gamma\left(\frac{3}{2}\right)^2$。

[Wallis 自己的启发式"证明"可以在 D. J. Struik 的*Source Book in Mathematics* (Harvard University Press, 1969), 244 ~ 253 中找到。]

19. 变量 $t = mt$ 的改变,分部积分和归纳法。

20. [为完全计,我们证明所述的不等式。从易于验证的不等式 $1 + x \leqslant e^x$ 开始;置 $x = \pm t/n$ 并取 n 次幂以得到 $(1 \pm t/n)^n \leqslant e^{\pm t}$。因此由习题 1.2.1-9,得到 $e^{-t} \geqslant (1 - t/n)^n = e^{-t}(1 - t/n)^n e^t \geqslant e^{-t}(1 - t/n)^n(1 + t/n)^n = e^{-t}(1 - t^2/n^2) \geqslant e^{-t}(1 - t^2/n)$。]

现在,给定的积分减 $\Gamma_m(x)$ 是

$$\int_m^\infty e^{-t} t^{x-1} dt + \int_0^m \left(e^{-t} - \left(1 - \frac{t}{m}\right)^m\right) t^{x-1} dt$$

当 $m \to \infty$ 时,第一个积分趋于 0,因为对于充分大的 t, $t^{x-1} < e^{t/2}$;而第二个积分小于

$$\frac{1}{m}\int_0^m t^{x+1} e^{-t} dt < \frac{1}{m}\int_0^\infty t^{x+1} e^{-t} dt \to 0$$

21. 如果 $c(n, j, k_1, k_2, \cdots)$ 表示相应的系数,则我们由微分求得

$$c(n+1, j, k_1, \cdots) = c(n, j-1, k_1-1, k_2, \cdots) + (k_1+1)c(n, j, k_1+1, k_2-1, k_3, \cdots) + (k_2+1)c(n, j, k_1, k_2+1, k_3-1, k_4, \cdots) + \cdots$$

等式 $k_1 + k_2 + \cdots = j$ 和 $k_1 + 2k_2 + \cdots = n$ 在这个归纳关系中被保持。我们能容易地由出现于 $c(n+1, j, k_1, \cdots)$ 的等式右边中每一项,分解出因子 $n!/(k_1!(1!)^{k_1} k_2!(2!)^{k_2}\cdots)$,而且我们保持有 $k_1 + 2k_2 + 3k_3 + \cdots = n + 1$。(在这个证明中,假定有无限多个 k 是方便的,尽管很明显 $k_{n+1} = k_{n+2} = \cdots = 0$。)

刚才给出的解答,利用了标准技术,但它并没有难出为什么这个公式有此形式之令人满意的解释,也没有说明首先是如何发现的。让我们使用由 H. S. Wall 提议的组合论证[*Bull. Amer.*

Math. Soc. **44** (1938),395 ~ 398]来考察这个问题。为方便起见，写 $w_j = D_u^j w$ ，$u_k = D_x^k u$。于是 $D_x(w_j) = w_{j+1}u_1$ 和 $D_x(u_k) = u_{k+1}$。由这两个规则以及求乘积导数的规则，我们求得

$D_x^1 w = w_1 u_1$

$D_x^2 w = (w_2 u_1 u_1 + w_1 u_2)$

$D_x^3 w = ((w_3 u_1 u_1 u_1 + w_2 u_2 u_1 + w_2 u_1 u_2) + (w_2 u_1 u_2 + w_1 u_3))$，等等。

类似地，我们可以建立对应的集合分划的表示如下：

$\mathscr{D}^1 = \{1\}$

$\mathscr{D}^2 = (\{2\}\{1\} + \{2,1\})$

$\mathscr{D}^3 = ((\{3\}\{2\}\{1\} + \{3,2\}\{1\} + \{2\}\{3,1\} + (\{3\}\{2,1\} + \{3,2,1\}))$，等等。

形式上，如果 $a_1 a_2 \cdots a_j$ 是集合 $\{1,2,\cdots,n-1\}$ 的一个分划，则定义

$$\mathscr{D}a_1 a_2 \cdots a_j = \{n\} a_1 a_2 \cdots a_j + (a_1 \cup \{n\}) a_2 \cdots a_j + a_1 (a_2 \cup \{n\}) \cdots a_j + \cdots + a_1 a_2 \cdots (a_j \cup \{n\})$$

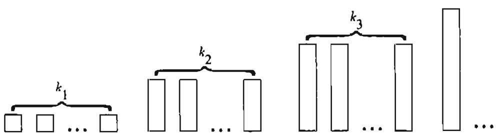
这个规则确切地平行于规则

$$D_x(w_j u_{r_1} u_{r_2} \cdots u_{r_j}) = w_{j+1} u_1 u_{r_1} u_{r_2} \cdots u_{r_j} + w_j u_{r_1+1} u_{r_2} \cdots u_{r_j} + w_j u_{r_1} u_{r_2+1} \cdots u_{r_j} + \cdots + w_j u_{r_1} u_{r_2} \cdots u_{r_j+1}$$

如果我们令项 $w_j u_{r_1} u_{r_2} \cdots u_{r_j}$ 对应于具有 r_t 个元素在 a_t 中的分划 $a_1 a_2 \cdots a_j$，$1 \leqslant t \leqslant j$ 的话。所以有从 $\mathscr{D}^n$ 到 $D_x^n w$ 上的自然映射，进而容易看出，$\mathscr{D}^n$ 恰好包括集合 $\{1,2,\cdots,n\}$ 的每个分划恰一次(参照习题 1.2.6-64)。

由这些观察，我们发现，如果我们合并 $D_x^n w$ 中的同类项，就得到项 $c(k_1,k_2,\cdots) w_j u_1^{k_1} u_2^{k_2} \cdots$ 的和，其中 $j = k_1 + k_2 + \cdots$ 和 $n = k_1 + 2k_2 + \cdots$，而 $c(k_1,k_2,\cdots)$ 是把 $\{1,2,\cdots,n\}$ 分划成 j 个子集，使得有 k_t 个子集有 t 个元素的分划数。

剩下就是来计算这分划数。考虑容量 t 的 k_t 个盒子的阵列：



放置 n 个不同的元素于这些盒子内的放法种数，是多项式系数

$$\binom{n}{1,1,\cdots,1,2,2,\cdots,2,3,3,\cdots,3,4,\cdots} = \frac{n!}{1!^{k_1} 2!^{k_2} 3!^{k_3} \cdots}$$

为了得到 $c(k_1,k_2,k_3,\cdots)$，应以 $k_1!\,k_2!\,k_3!\cdots$ 来除这个数，因为在每个 k_t 的组中，诸盒子是不能互相区别的；它们可以以 $k_t!$ 种方法来进行排列而不影响这集合的分划。

Arbogast 原来的证明[*Du Calcul des Dérivations* (Strasbourg: 1800) § 52]是基于以下事实：$D_x^k u / k!$ 是 $u(x+z)$ 中 z^k 的系数，而 $D_u^j w / j!$ 是 $w(u+y)$ 中 y^j 的系数，因此在 $w(u(x+z))$ 中 z^n 的系数为

$$\frac{D_x^n w}{n!} = \sum_{j=0}^{n} \frac{D_u^j w}{j!} \sum_{\substack{k_1+k_2+\cdots+k_n=j \\ k_1+2k_2+\cdots+nk_n=n \\ k_1,k_2,\cdots,k_n \geqslant 0}} \frac{j!}{k_1!\,k_2!\cdots k_n!} \left(\frac{D_x^1 u}{1!}\right)^{k_1} \left(\frac{D_x^2 u}{2!}\right)^{k_2} \cdots \left(\frac{D_x^n u}{n!}\right)^{k_n}$$

他的公式被遗忘了许多年，后来由 F. Faà di Bruno 独立地重新发现[*Quarterly J. Math.* **1** (1857), 359 ~ 360]，他还发现，也可把它表达为一个行列式

$$D_x^n=\det\begin{pmatrix}\binom{n-1}{0}u_1 & \binom{n-1}{1}u_2 & \binom{n-1}{2}u_3 & \cdots & \binom{n-1}{n-2}u_{n-1} & \binom{n-1}{n-1}u_n\\ -1 & \binom{n-2}{0}u_1 & \binom{n-2}{1}u_2 & \cdots & \binom{n-2}{n-3}u_{n-2} & \binom{n-2}{n-2}u_{n-1}\\ 0 & -1 & \binom{n-3}{0}u_1 & \cdots & \binom{n-3}{n-4}u_{n-3} & \binom{n-3}{n-3}u_{n-2}\\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & 0 & \cdots & -1 & \binom{0}{0}u_1\end{pmatrix}$$

其中 $u_j=(D_x^j u)D_u$；这个等式的两边是要应用于 w 上的微分算子。关于 Arbogast 的公式向若干个变量的函数的推广，以及对于其它有关著作的参考文献目录，请见 I. J. Good 的论文，*Annals of Mathematical Statistics* **32** (1961)，540 ~ 541。

22. 关于 $\lim_{n\to\infty}(n+x)!/(n!\,n^x)=1$ 的假设，对整数 x 是正确的；例如，如果 x 为正，则这个量是 $(1+1/n)(1+2/n)\cdots(1+x/n)$，它确实趋于 1。如果我们还假定 $x!=x(x-1)!$，则这个假设就把我们直接引向结论

$$1=\lim_{n\to\infty}\frac{(n+x)!}{n!\,n^x}=x!\lim_{x\to\infty}\frac{(x+1)\cdots(x+n)}{n!\,n^x}$$

这就等价于正文所给出的定义。

23. 由(13)和(15)，$z(-z)!\,\Gamma(z)=\lim_{m\to\infty}\prod_{n=1}^{m}(1-z/n)^{-1}(1+z/n)^{-1}$。

24. $n^n/n!=\prod_{k=1}^{n-1}(k+1)^k/k^k\leqslant\prod_{k=1}^{n-1}e$；$n!/n^{n+1}=\prod_{k=1}^{n-1}k^{k+1}/(k+1)^{k+1}\leqslant\prod_{k=1}^{n-1}e^{-1}$。

25. $x^{\underline{m+n}}=x^{\underline{m}}(x+m)^{\underline{n}}$；$x^{\overline{m+n}}=x^{\overline{m}}(x-m)^{\overline{n}}$。由(21)，当 m 和 n 是非整数时，这些定律也成立。

1.2.6 小节

1. n，因为每个组合丢掉一项。

2. 1。从空集什么也不选的方法只有一种。

3. $\binom{52}{13}$。实际的数是 635013559600。

4. $2^4\cdot5^2\cdot7^2\cdot17\cdot23\cdot41\cdot43\cdot47$。

5. $(10+1)^4=10000+4(1000)+6(100)+4(10)+1$。

6.
$r=-3$:	1	−3	6	−10	15	−21	28	−36	⋯
$r=-2$:	1	−2	3	−4	5	−6	7	−8	⋯
$r=-1$:	1	−1	1	−1	1	−1	1	−1	⋯

7. $\lfloor n/2\rfloor$；或者，$\lceil n/2\rceil$。由(3)显见，对于较小的值，二项式系数是严格地递增的，而后递减为 0。

8. 每行中的非 0 项，从左到右来读与从右到左来读是相同的。

9. 如果 n 为正或 0，则值为 1；如果 n 为负，则值为 0。

10. a)，b)和 f)可直接由 e)得出；c)和 d)可由 a)，b)和等式(9)得出。因此只需证明 e)就足够了。把$\binom{n}{k}$当做由等式(3)给出的分数，且在分子和分母中有诸因子出现。前 $k \bmod p$ 个因子在分母中没有 p，而在分子和分母中这些项显然同余于

$$\binom{n \bmod p}{k \bmod p}$$

相应的诸项，这相差 p 的倍数。(当处理 p 之非倍数时，我们可以在分子和分母两者中都进行模 p 的操作，因为如果 $a \equiv c$ 和 $b \equiv d$，以及 a/b，c/d 都是整数，则 $a/b \equiv c/d$。)于是还剩下 $k - k \bmod p$个因子，它们落入到各有 p 个连续值的$\lfloor k/p \rfloor$个组当中。每个组恰包含一个 p 的倍数；在一个组中的其它 $p-1$ 个因子模 p 同余于$(p-1)!$，因此在分子和分母中相消。剩下来研究在分子和分母中的 p 的$\lfloor k/p \rfloor$倍数。我们对这些中的每一个除以 p 因而剩下二项式系数

$$\binom{\lfloor (n - k \bmod p)/p \rfloor}{\lfloor k/p \rfloor}$$

如果 $k \bmod p \leqslant n \bmod p$，如要求的那样，此值等于

$$\binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor}$$

而且如果 $k \bmod p > n \bmod p$，则其它因子$\binom{n \bmod p}{k \bmod p}$为零，所以一般说来这个公式成立。[*American J. Math.* **1** (1878)，229 ~ 230；也见 N. J. Fine，*AMM* **54** (1947)，589 ~ 592。]

11. 如果 $a = a_rp^r + \cdots + a_0$，$b = b_rp^r + \cdots + b_0$，$a + b = c_rp^r + \cdots + c_0$，则 n 的值(按照习题 1.2.5-12 和等式(5))是

$$(a_0 + \cdots + a_r + b_0 + \cdots + b_r - c_0 - \cdots - c_r)/(p-1)$$

进位使 c_j 减少 p 并使 c_{j+1}增 1，在这个公式中引起 +1 的净变化。[对于 q 多项式和斐波那契系数类似的结果成立；请见 Knuth 和 Wilf，*Crelle* **396** (1989)，212 ~ 219。]

12. 由上两题中之任一题，n 必须比 2 的幂小 1。更一般地，$\binom{n}{k}$绝不能为素数 p 所整除，$0 \leqslant k \leqslant n$，当且仅当 $n = ap^m - 1$，$1 \leqslant a < p$，$m \geqslant 0$。

14. $24\binom{n+1}{5} + 36\binom{n+1}{4} + 14\binom{n+1}{3} + \binom{n+1}{2} =$

$$\frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30} = \frac{n(n+1)(n+\frac{1}{2})(3n^2+3n-1)}{15}$$

15. 用归纳法和等式(9)。

17. 我们可以假定 r，s 是正整数。而且对于所有 x，

$$\sum_n \binom{r+s}{n} x^n = (1+x)^{r+s} = \sum_k \binom{r}{k} x^k \sum_m \binom{s}{m} x^m =$$

$$\sum_k \binom{r}{k} x^k \sum_n \binom{s}{n-k} x^{n-k} = \sum_n \left(\sum_k \binom{r}{k} \binom{s}{n-k} \right) x^n$$

所以 x^n 的系数一定相等。

21. 左边是次数不大于 n 的多项式,右边是 $m+n+1$ 次多项式。多项式已经在 $n+1$ 个点上一致,但这并不足以证明它们相等。[事实上,正确的一般公式为

$$\sum_{k=0}^{r}\binom{r-k}{m}\binom{s+k}{n}=\binom{r+s+1}{m+n+1}-\sum_{k=0}^{m}\binom{r+1}{k}\binom{s}{m+n+1-k}$$

这里 m,n 和 r 为非负整数。]

22. 假定 $n>0$。第 k 项是 r 乘

$$\frac{1}{n!}\binom{n}{k}\prod_{0<j<k}(r-tk-j)\prod_{0\leq j<n-k}(n-1-r+tk-j)=$$
$$\frac{(-1)^{k-1}}{n!}\binom{n}{k}\prod_{0<j<k}(-r+tk+j)\prod_{k\leq j<n}(-r+tk+j)$$

而且这两个乘积给出 k 的 $n-1$ 次多项式,所以由等式(34),对 k 求和为零。

24. 证明通过对 n 用归纳法来进行。如果 $n\leq 0$,则恒等式是明显的。如果 $n>0$,则我们通过对整数 $m\geq 0$ 用归纳法,并利用上边的两道习题和它们对于 $n-1$ 的正确性,证明它对于 $(r,n-r+nt+m,t,n)$ 成立。这就对于无穷多的 s 确立了 (r,s,t,n) 的恒等性,它对所有的 s 成立,因为两边都是 s 的多项式。

25. 利用比率测试和对于大的 k 值的直截了当的估计,我们可以证明收敛性。(或者利用复变理论,我们知道这函数在 $x=1$ 的邻域中是解析的。)我们有

$$1=\sum_{k,j}(-1)^j\binom{k}{j}\binom{r-jt}{k}\frac{r}{r-jt}w^k=\sum_j(-1)^j\frac{r}{r-jt}\sum_k\binom{k}{j}\binom{r-jt}{k}w^k=$$
$$\sum_j\frac{(-1)^jr}{r-jt}\sum_k(r-jt)\binom{r-jt-j}{k-j}w^k=\sum_j(-1)^jA_j(r,t)(1+w)^{r-jt-j}w^j$$

现在令 $x=1/(1+w)$, $z=-w/(1+w)^{1+t}$。这个证明是 H. W. Gould 给出的[*AAM* **63** (1956), 84~91]。也可参看在习题 2.3.4-33 和 4.7-22 中的更一般的公式。

26. 我们将通过形如

$$\sum_j(-1)^j\binom{k}{j}\binom{r-jt}{k}=t^k$$

的恒等式(35)开始,并如在习题 25 中那样进行。另一种方法是把该习题中的公式对 z 求微分,我们得到

$$\sum_k kA_k(r,t)z^k=z\frac{\mathrm{d}(x^r)}{\mathrm{d}z}=\frac{(x^{t+1}-x^t)rx^r}{(t+1)x^{t+1}-tx^t}$$

由此可得到

$$\sum_k\left(1-\frac{t}{r}k\right)A_k(r,t)z^k$$

的值。

27. 对于等式(26),以 x^s 的级数来乘 $x^{r+1}/((t+1)x-t)$ 的级数,得到 $x^{r+s+1}/((t+1)x-t)$ 的级数,其中 z 的系数可以等于由 $x^{(r+s)+1}/((t+1)x-t)$ 产生的级数之系数。

28. 以 $f(r,s,t,n)$ 来表示左边,通过考虑恒等式

$$\sum_k \binom{r+tk}{k}\binom{s-tk}{n-k}\frac{r}{r+tk} + \sum_k \binom{r+tk}{k}\binom{s-tk}{n-k}\frac{tk}{r+tk} = f(r,s,t,n)$$

我们求得

$$\binom{r+s}{n} + tf(r+t-1, s-t, t, n-1) = f(r,s,t,n)$$

29. $(-1)^k\binom{n}{k}/n! = (-1)^k/(k!(n-k)!) = (-1)^n/\prod_{\substack{0\le j\le n\\ j\ne k}}(k-j)$。

30. 应用(7),(6)和(19)求得

$$\sum_{k\ge 0}\binom{-m-2k-1}{n-m-k}\binom{2k+1}{k}\frac{(-1)^{n-m}}{2k+1}$$

现在我们可应用等式(26)连同$(r,s,t,n) = (1, m-2n-1, -2, n-m)$,得到

$$(-1)^{n-m}\binom{-m}{n-m} = \binom{n-1}{n-m}$$

当 n 为正时,这个结果与我们以前的公式相同,但当 $n=0$ 时,我们已经得到的答案是正确的,而 $\binom{n-1}{m-1}$则不然。我们的推导还有进一步的收获,因为对于 $n\ge 0$ 和所有整数 m,答案 $\binom{n-1}{n-m}$是正确的。

31.[这个和式首先由 J. F. Pfaff 以一种相近的形式得到,*Nova Acta Acad. Scient. Petr*. **11** (1797),38~57。]我们有

$$\sum_k\sum_j\binom{m-r+s}{k}\binom{n+r-s}{n-k}\binom{r}{m+n-j}\binom{k}{j} =$$

$$\sum_k\sum_j\binom{m-r+s}{j}\binom{n+r-s}{n-k}\binom{r}{m+n-j}\binom{m-r+s-j}{k-j} =$$

$$\sum_j\binom{m-r+s}{j}\binom{r}{m+n-j}\binom{m+n-j}{n-j}$$

把$\binom{m+n-j}{n-j}$变为$\binom{m+n-j}{m}$并且再次应用(20),我们得到

$$\sum_j\binom{m-r+s}{j}\binom{r}{m}\binom{r-m}{n-j} = \binom{r}{m}\binom{s}{n}$$

32. 在(44)中以 $-x$ 代替 x。

33,34. [*Giornale di Mat. Battaglini* **31** (1893),291~313;**33** (1895),179~182。]我们有 $x^{\overline{n}} = n!\cdot\binom{x+n+1}{n}$。这个等式因此可以变换成

$$\binom{x+y+n-1}{n} = \sum_k\binom{x+(1-z)k}{k}\binom{y-1+nz+(n-k)(1-z)}{n-k}\frac{x}{x+(1-z)k}$$

这就是(26)中的情形。类似地,$(x+y)^{\underline{n}}=\sum_k\binom{n}{k}x(x-kz-1)^{\underline{k-1}}(y+kz)^{\underline{n-k}}$,这是 Rothe 的等价的公式[*Formulæ de Serierum Reversione* (Leipzig: 1793), 18]。

35. 例如,我们证明第一个公式:

$$\sum_k(-1)^{n+1-k}\left(n\begin{bmatrix}n\\k\end{bmatrix}+\begin{bmatrix}n\\k-1\end{bmatrix}\right)x^k=-nx^{\underline{n}}+xx^{\underline{n}}=x^{\underline{n+1}}$$

36. 由(13),假定 n 是非负整数,我们分别得到 2^n 和 δ_{n0}。

37. 当 $n>0$ 时为 2^{n-1}。(奇数项和偶数项相抵消,因此每一个等于总和之半。)

38. 令 $\omega=e^{2\pi i/m}$,则

$$\sum_{0\le j<m}(1+\omega^j)^n\omega^{-jk}=\sum_t\sum_{0\le j<m}\binom{n}{t}\omega^{j(t-k)}$$

现在

$$\sum_{0\le j<m}\omega^{rj}=m[r\equiv 0(\text{modulo } m)]$$

(这是几何级数的和),所以右边的和为 $m\sum_{t \bmod m=k}\binom{n}{t}$。左边原来的和为

$$\sum_{0\le j<m}(\omega^{-j/2}+\omega^{j/2})^n\omega^{j(n/2-k)}=\sum_{0\le j<m}\left(2\cos\frac{j\pi}{m}\right)^n\omega^{j(n/2-k)}$$

因为已知这个量是实数,我们可以取实数部分并得到所述公式。[见 *Crelle* **11** (1834), 353 ~ 355。]

在*CMath* 中,习题 5.75 和 6.57 讨论了 $m=3$ 和 $m=5$ 的情况的特殊性质。

39. $n!$; $\delta_{n0}-\delta_{n1}$。(在第二个三角形中的行之和并不那么简单;我们将发现(习题 64),$\sum_k\begin{Bmatrix}n\\k\end{Bmatrix}$是把 n 个元素的集合分划成不相交集合的方式数,它就是在$\{1,2,\cdots,n\}$上的等价关系的个数。)

40. (c)的证明:通过分部积分,

$$B(x+1,y)=\left.\frac{t^x(1-t)^y}{y}\right|_0^1+\frac{x}{y}\int_0^1 t^{x-1}(1-t)^y\,dt$$

现在使用(b)。

41. 不论 m 取遍整数值与否(由单调性),当 $m\to\infty$ 时,$m^xB(x,m+1)\to\Gamma(x)$。因此,$(m+y)^xB(x,m+y+1)\to\Gamma(x)$,而且$(m/(m+y))^x\to 1$。

42. 如果这是按照习题 41(b)定义的,则为 $1/((r+1)B(k+1,r-k+1))$。一般说来,当 z 和 w 是任意的复数时,我们定义

$$\binom{z}{w}=\lim_{\zeta\to z}\lim_{\omega\to w}\frac{\zeta!}{\omega!(\zeta-\omega)!},\ \text{其中}\ \zeta!=\Gamma(\zeta+1)$$

当 z 是负整数和 w 不是整数时,这个值是无穷的。

通过这个定义,对于所有复数 n 和 k,对称条件(6)成立,但当 n 是负整数和 k 是整数时除外;等式(7),(9)和(20)永远是成立的,尽管有时它们可能取不定式的形式,例如 $0\cdot\infty$ 或 $\infty+\infty$。等式(17)变成

$$\binom{z}{w}=\frac{\sin\pi(w-z-1)}{\sin\pi z}\binom{w-z-1}{w}$$

我们甚至能够推广二项式定理(13)和范德蒙德卷积(21),得到$\sum_k\binom{r}{\alpha+k}z^{\alpha+k}=(1+z)^r$和$\sum_k\binom{r}{\alpha+k}\binom{s}{\beta-k}=\binom{r+s}{\alpha+\beta}$;假定对复数幂适当地定义,每当级数收敛时,对于所有复数r,s,z,α和β,这些公式都成立。[请见 L. Ramshaw, *Inf. Proc. Letters* **6** (1977), 223~226。]

43. $\int_0^1 dt/(t^{1/2}(1-t)^{1/2})=2\int_0^1 du/(1-u^2)^{1/2}=2\arcsin u\Big|_0^1=\pi$。

45. 对于充分大的 r 的值,我们有

$$\frac{1}{k\Gamma(k)}\sqrt{\frac{r}{r-k}}\,\frac{1}{e^k}\,\frac{(1-k/r)^k}{(1-k/r)^r}\to\frac{1}{\Gamma(k+1)}$$

46. $\sqrt{\frac{1}{2\pi}\left(\frac{1}{x}+\frac{1}{y}\right)}\left(1+\frac{y}{x}\right)^x\left(1+\frac{x}{y}\right)^y$,以及$\binom{2n}{n}\approx 4^n/\sqrt{\pi n}$。

47. 当 $k\leq 0$ 时每个量都是 δ_{k0},而且当用 $k+1$ 代替 k 时,每个量都被乘以$(r-k)(r-\frac{1}{2}-k)/(k+1)^2$。当 $r^2=-\frac{1}{2}$时,这就意味着$\binom{-1/2}{k}=(-1/4)^k\binom{2k}{k}$。

48. 这可以用归纳法证明,利用下列事实,即当 $n>0$ 时,

$$0=\sum_k\binom{n}{k}(-1)^k=\sum_k\binom{n}{k}\frac{(-1)^k k}{k+x}+\sum_k\binom{n}{k}\frac{(-1)^k x}{k+x}$$

或者,我们有

$$\mathrm{B}(x,n+1)=\int_0^1 t^{x-1}(1-t)^n dt=\sum_k\binom{n}{k}(-1)^k\int_0^1 t^{x+k-1}dt$$

(事实上,当级数收敛时,对于非整数 n,所述的和也等于 $\mathrm{B}(x,n+1)$。)

49. $\binom{r}{m}=\sum_k\binom{r}{k}\binom{-r}{m-2k}(-1)^{m+k}$,$m$ 为整数(见习题 17)。

50. 第 k 个被加项是$\binom{n}{k}(-1)^{n-k}(x-kz)^{n-1}x$。应用等式(34)。

51. 右边是

$$\sum_k\binom{n}{n-k}x(x-kz)^{k-1}\sum_j\binom{n-k}{j}(x+y)^j(-x+kz)^{n-k-j}=$$

$$\sum_j\binom{n}{j}(x+y)^j\sum_k\binom{n-j}{n-j-k}x(x-kz)^{k-1}(-x+kz)^{n-k-j}=$$

$$\sum_{j\leq n}\binom{n}{j}(x+y)^j 0^{n-j}=(x+y)^n$$

同样的手段也可用来证明 Torelli 和式(习题 34)。

阿贝尔公式另一个简洁的证明来自于以下的事实,即它很容易转换成由习题 2.3.4.4-29

所导出的更加对称的恒等式

$$\sum_k \binom{n}{k} x(x+kz)^{k-1} y(y+(n-k)z)^{n-k-1} = (x+y)(x+y+nz)^{n-1}$$

阿贝尔定理甚至已经由 A. Hurwitz 进一步推广[*Acta Mathematica* **26** (1902),199～203]如下：

$$\sum x(x+\epsilon_1 z_1+\cdots+\epsilon_n z_n)^{\epsilon_1+\cdots+\epsilon_n-1}(y-\epsilon_1 z_1-\cdots-\epsilon_n z_n)^{n-\epsilon_1-\cdots-\epsilon_n} = (x+y)^n$$

其中的求和是对于 $\epsilon_1,\cdots,\epsilon_n=0$ 或 1 的所有 2^n 个独立的选择进行的。这是关于 $x,y,z_1,\cdots,z_n$ 的恒等式,而阿贝尔公式是当 $z_1=z_2=\cdots=z_n$ 时的特殊情况。Hurwitz 的公式由习题 2.3.4.4-30 的结果得出。

52. $\sum_{k\geq 0}(k+1)^{-2}=\pi^2/6$。[M. L. J. Hautus 发现,每当 $z\neq 0$ 时,对于所有复数 x,y,z,n 这个求和绝对收敛,因为对于很大的 k 的诸项总有 $1/k^2$ 的数量级。在有界区域中这个收敛是一致的,所以我们可以对这个级数逐项求微分。如果 $f(x,y,n)$ 是当 $z=1$ 时的和式的值,我们求得 $(\partial/\partial y)f(x,y,n)=nf(x,y,n-1)$ 和 $(\partial/\partial x)f(x,y,n)=nf(x-1,y+1,n-1)$。这些公式同 $f(x,y,n)=(x+y)^n$ 是一致的;但实际上后一个等式似乎很少成立,如果说有的话,除非这个和是有限的。其次,关于 z 的导数几乎总是非零。]

53. 对于(b),在(a)的结果中置 $r=\frac{1}{2}$ 和 $s=-\frac{1}{2}$。

54. 棋盘模式中插入负号如下：

$$\begin{pmatrix} 1 & -0 & 0 & -0 \\ -1 & 1 & -0 & 0 \\ 1 & -2 & 1 & -0 \\ -1 & 3 & -3 & 1 \end{pmatrix}$$

这就等于以 $(-1)^{i+j}$ 来乘 a_{ij}。由等式(33),这结果是所求的逆矩阵。

55. 像在上题中那样,在三角形矩阵内插入负号,得到另一个的逆矩阵(等式(47))。

56. 210 310 320 321 410 420 421 430 431 432 510 520 521 530 531 532 540 541 542 543 610。通过固定 a,b 和 c 跑遍 a 件事物每次取 2 件的组合;通过固定 a 和 b,c 跑遍 b 件事物每次取 1 件的组合。

类似地,以 $a>b>c>d\geq 0$,可以表达形如 $n=\binom{a}{4}+\binom{b}{3}+\binom{c}{2}+\binom{d}{1}$ 的所有的数;这个序列以 3210 4210 4310 4320 4321 5210 5310 5320 …开始。通过一种"贪婪"的方法,我们可以找出组合表示,首先选择最大可能的 a,然后对于 $n-\binom{a}{4}$ 找出最大可能的 b,等等。[7.2.1.3 小节讨论此表示的进一步性质。]

58. 用归纳法,由于

$$\binom{n}{k}_q = \binom{n-1}{k}_q + \binom{n-1}{k-1}_q q^{n-k} = \binom{n-1}{k}_q q^k + \binom{n-1}{k-1}_q$$

由此推出(21)的 q 推广是

$$\sum_k \binom{r}{k}_q \binom{s}{n-k}_q q^{(r-k)(n-k)} = \sum_k \binom{r}{k}_q \binom{s}{n-k}_q q^{(s-n+k)k} = \binom{r+s}{n}_q$$

而恒等式 $1-q^t=-q^t(1-q^{-t})$ 使得很容易把(17)推广成

$$\binom{r}{k}_q=(-1)^k\binom{k-r-1}{k}_q q^{kr-k(k-1)/2}$$

q 多项式的系数出现在许多不同的应用中;例如,参见 5.1.2 小节和作者在*J. Combinatorial Theory* **A10** (1971),178 ~ 180 中的点评。

有用的事实:当 n 是非负整数时,$\binom{n}{k}_q$ 是具有非负整数系数的 q 的 $k(n-k)$ 次多项式,而且它满足反射律

$$\binom{n}{k}_q=\binom{n}{n-k}_q=q^{k(n-k)}\binom{n}{k}_{q^{-1}}$$

如果 $|q|<1$ 和 $|x|<1$,则当 n 为任意实数时,如果我们以 $\prod_{k\geq 0}((1+q^kx)/(1+q^{n+k}x))$ 代替左边,则 q 多项式定理成立。幂级数的性质使得仅当 n 是正整数时才有必要验证这一点,因为我们可以置 $q^n=y$;对于无穷多个 y 的值已经验证了恒等式。现在我们可以对 q 多项式定理中的上标取负得到由柯西[*Comptes Rendus Acad. Sci.* **17** (Paris, 1843),523 ~ 531]和 Heine [*Crelle* **34** (1847),285 ~ 328]给出的公式

$$\prod_{k\geq 0}\frac{(1-q^{k+r+1}x)}{(1-q^kx)}=\sum_k\binom{-r-1}{k}_q q^{k(k-1)/2}(-q^{r+1}x)^k=\sum_k\binom{k+r}{k}_q x^k$$

关于进一步的信息,参见 G. Gasper 和 M. Rahman,*Basic Hypergeometric Series* (Cambridge Univ. Press,1990)。

59. $(n+1)\binom{n}{k}-\binom{n}{k+1}$。

60. $\binom{n+k-1}{k}$。这个公式容易记住,因为它是

$$\frac{n(n+1)\cdots(n+k-1)}{k(k-1)\cdots 1}$$

与等式(2)相似,只是分子中的数往上升而不是往下降。欲证这个公式,有一个乖巧的方法,就是请注意,我们要计算对于关系 $1\leq a_1\leq a_2\leq\cdots\leq a_k\leq n$ 的整数解 $(a_1,\cdots,a_k)$ 的个数。这关系就等同于 $0<a_1<a_2+1<\cdots<a_k+k-1<n+k$;而且对于

$$0<b_1<b_2<\cdots<b_k<n+k$$

的解的个数,是由集合 $\{1,2,\cdots,n+k-1\}$ 中选取 k 件不同的事物之选择个数。(这一技巧是由 H. F. Scherk 给出的,*Crelle* **3** (1828),97;说来奇怪,W. A. Förstemann 也在同一杂志上给出它,**13** (1835),237,而他却说“人们几乎都以为这必然是很久以前就已经知道了,但是我哪儿也没有找到它,尽管我已经考查了这方面的许多著作”。)

61. 如果 a_{mn} 是所要求的量,由(46)和(47)我们有 $a_{mn}=na_{m(n-1)}+\delta_{mn}$。因此答案是 $[n\geq m]\cdot n!/m!$。通过(56)的反演也很容易得到同一公式。

62. 利用习题 31 的恒等式,通过 $(m,n,r,s,k)\leftarrow(m+k,l-k,m+n,n+l,j)$ 并重新安排阶乘的符号得到

$$\sum_k (-1)^k \binom{l+m}{l+k}\binom{m+n}{m+k}\binom{n+l}{n+k} =$$

$$\sum_{j,k} (-1)^k \binom{l+m}{l+k}\binom{l+k}{j}\binom{m-k}{l-k-j}\binom{m+n+j}{m+l} =$$

$$\sum_{j,k} (-1)^k \binom{2l-2j}{l-j+k} \frac{(m+n+j)!}{(2l-2j)!\,j!\,(m-l+j)!\,(n+j-l)!}$$

对 k 之求和现在为 0,除非 $j=l$。

这个恒等式的 $l=m=n$ 之情形,是由 A. C. Dixon 发表的[*Messenger of Math.* **20** (1891),79~80],12 年后他确立了一般的情况[*Proc. London Math. Soc.* **35** (1903),285~289]。然而,与此同时,L. J. Rogers 发表了更一般得多的公式[*Proc. London Math. Soc.* **26** (1895),15~32, §8]。也见 P. A. MacMahon 的论文,*Quarterly Journal of Pure and Applied Math.* **33** (1902),274~288,以及 John Dougall,*Proc. Edinburgh Math. Society* **25** (1907),114~132。对应的 q 多项式的恒等式是

$$\sum_k \binom{m-r+s}{k}_q \binom{n+r-s}{n-k}_q \binom{r+k}{m+n}_q q^{(m-r+s-k)(n-k)} = \binom{r}{m}_q \binom{s}{n}_q$$

$$\sum_k (-1)^k \binom{l+m}{l+k}_q \binom{m+n}{m+k}_q \binom{n+l}{n+k}_q q^{(3k^2-k)/2} = \frac{(l+m+n)!_q}{l!_q m!_q n!_q}$$

其中 $n!_q = \prod_{k=1}^n (1+q+\cdots+q^{k-1})$。

63. 请见*CMath*,习题 5.83 和 5.106。

64. 设 $f(n,m)$ 是把 $\{1,2,\cdots,n\}$ 分成 m 个部分的分划数。显然,$f(1,m)=\delta_{1m}$。如果 $n>1$,则这种分划有两种:(a)元素 n 单独地形成该分划的一个集合;共有 $f(n-1,m-1)$ 种方式来构造像这样的分划;(b)元素 n 与另外的元素一起出现,共有 m 种方式来把 n 插入到 $\{1,2,\cdots,n-1\}$ 的任何 m 分划中,因此有 $mf(n-1,m)$ 种方式来构造像这样的分划。于是得出结论 $f(n,m)=f(n-1,m-1)+mf(n-1,m)$,而且由归纳法 $f(n,m)=\left\{ {n \atop m} \right\}$。

65. 请见*AMM* **99** (1992),410~422。

66. 令 $X=\binom{x}{n}$,$\underline{X}=\binom{x}{n-1}=\frac{n}{x-n+1}X$,$\overline{X}=\binom{x}{n+1}=\frac{n}{x-n+1}X$,类似地定义 Y 和 Z。我们可以假定 $y>n-1$ 固定,使得 x 是 z 的函数。

令 $F(z)=\overline{X}-\overline{Y}-\overline{Z}$,并假定对某个 $z>n-2$,$F(z)=0$,可以证明 $F'(z)<0$;因此 $z=y$ 一定是大于等于 $n-2$ 的惟一根,这就证明了第 2 个不等式。因为 $F(z)=\frac{x-n}{n+1}(Y+Z)-\frac{y-n}{n+1}Y-\frac{z-n+1}{n}Z=0$ 且 $x>y$ 且 $Y,Z>0$,故一定有 $\frac{x-n}{n+1}<\frac{z-n+1}{n}$。置 $X'=\mathrm{d}X/\mathrm{d}x$ 且 $Z'=\mathrm{d}Z/\mathrm{d}z=\mathrm{d}X/\mathrm{d}z$,则得到

$$\frac{X'}{X}=\frac{1}{x}+\frac{1}{x-1}+\cdots+\frac{1}{x-n+1}>\frac{n}{n+1}\left(\frac{1}{z}+\cdots+\frac{1}{z-n+2}\right)=\frac{n}{n+1}\frac{Z'}{Z}$$

这是因为 $\frac{x-n+1}{n+1}<\frac{z-n+2}{n},\cdots,\frac{x-1}{n+1}<\frac{z}{n}$。因此 $\mathrm{d}x/\mathrm{d}z=Z'/X'<\frac{n+1}{n}(Z/X)$,而且

$$F'(z)=\frac{X}{n+1}\frac{dx}{dz}+\frac{x-n}{n+1}Z'-\frac{Z}{n}-\frac{z-n+1}{n}Z'<\left(\frac{x-n}{n+1}-\frac{z-n+1}{n}\right)Z'<0$$

欲证明第一个不等式,可假定 $n>2$。此时若对某个 $z>n-2$, $\underline{X}=\underline{Y}+\underline{Z}$,则第 2 个不等式告诉我们 $z=y$。

[参见 L. Lovász, *Combinatorial Problems and Exercises* , Problem 13.31(a); R. M. Redheffer, *AMM* **103** (1996),62 ~ 64。]

67. 如果 $k>0$,习题 1.2.5 - 24 给出稍微更精确(但不大好记)的上界 $\binom{n}{k}=n^{\underline{k}}/k!\leqslant n^k/k!\leqslant\frac{1}{e}\left(\frac{ne}{k}\right)^k\leqslant\left(\frac{ne}{k+1}\right)^k$。对应的下界是 $\binom{n}{k}\geqslant\left(\frac{(n-k)e}{k}\right)^k\frac{1}{ek}$。

68. 设 $t_k=k\binom{n}{k}p^k(1-p)^{n+1-k}$;于是 $t_k-t_{k+1}=\binom{n}{k}p^k(1-p)^{n-k}(k-np)$。所以所述的和是 $\sum_{k<\lceil np\rceil}(t_{k+1}-t_k)+\sum_{k\geqslant\lceil np\rceil}(t_k-t_{k+1})=2t_{\lceil np\rceil}$。[De Moivre 在*Miscellanea Analytica* (1730),101 中,在 np 是整数的情况下指出了这个恒等式;H. Poincaré 在他的*Calcul des Probabilités* (1896),56 ~ 60 中证明了一般情况。关于这个恒等式的有趣历史以及关于形形色色的类似公式,请见 P. Diaconis 和 S. Zabell,*Statistical Science* **6** (1991),284 ~ 3002。]

1.2.7 小节

1. 0,1 和 3/2。
2. 以上界 $1/2^m$ 代替每项 $1/(2^m+k)$。
3. $H_{2^m-1}^{(r)}\leqslant\sum_{0\leqslant k<m}2^k/2^{kr}$;$2^{r-1}/(2^{r-1}-1)$是上界。
4. (b)和(c)。
5. 9.78760 60360 44382…。
6. 用归纳法和等式 1.2.6-(46)。
7. $T(m+1,n)-T(m,n)=1/(m+1)-1/(mn+1)\cdots-1/(mn+n)\leqslant1/(m+1)-(1/(mn+n)+\cdots+1/(mn+n))=1/(m+1)-n/(mn+n)=0$。当 $m=n=1$ 时出现极大值,而当 m 和 n 变得很大时,它趋于极小值。由等式(3),最大的下界为 γ,实际上这绝不被真正地达到。关于这个结果的一种推广,见于*AMM* **70** (1963),575 ~ 577。
8. 由斯特林近似公式,$\ln n!$ 近似于 $\left(n+\frac{1}{2}\right)\ln n-n+\ln\sqrt{2\pi}$;而 $\sum_{k=1}^{n}H_k$ 近似于 $(n+1)\cdot\ln n-n(1-\gamma)+(\gamma+\frac{1}{2})$;其差近似于 $\gamma n+\frac{1}{2}\ln n+.158$。
9. $-1/n$。
10. 把左边拆成两个和式,在第二个和式中把 k 变成 $k+1$。
11. 当 $n>0$ 时,$2-H_n/n-1/n$。
12. 1.000…正确到超过 300 位小数。
13. 像在定理 A 的证明中那样使用归纳法,或者使用微积分;对 x 求导数,而且在 $x=1$ 处求值。
14. 见 1.2.3 小节,例 2。第二个和式是 $\frac{1}{2}(H_{n+1}^2-H_{n+1}^{(2)})$。
15. $\sum_{j=1}^{n}(1/j)\sum_{k=j}^{n}H_k$ 可以通过正文中的公式来求和;答案为 $(n+1)H_n^2-(2n+1)H_n+2n$。
16. $H_{2n+1}-\frac{1}{2}H_{n-1}$。

17. 第一个解(初等的):取分母为$(p-1)!$,它是真正的分母的倍数但不是 p 的倍数,我们只需证明相应的分子$(p-1)!/1+(p-1)!/2+\cdots+(p-1)!/(p-1)$是 p 的倍数。取模 p,$(p-1)!/k\equiv(p-1)!\ k'$,其中 k'可通过关系 $kk' \bmod p=1$ 来确定。集合$\{1',2',\cdots,(p-1)'\}$恰是集合$\{1,2,\cdots,p-1\}$;因此分子同余于$(p-1)!\ (1+2+\cdots+p-1)\equiv 0$。

第二个解(高等解):由习题 4.6.2-6,我们有 $x^{\bar{p}}\equiv x^p-x$(modulo p);因此由习题 1.2.6-32,$\left[{p \atop k}\right]\equiv\delta_{kp}-\delta_{k1}$。现在应用习题 6。

当 $p>3$ 时,H_{p-1}的分子事实上是 p^2 的倍数;请见 Hardy 和 Wright,*An Introduction to the Theory of Numbers*, 7.8 节。

18. 若 $n=2^k m$,其中 m 为奇数,则和式等于 $2^{2k}m_1/m_2$,其中 m_1 和 m_2 都是奇数[*AMM* **67** (1960), 924 ~ 925]。

19. 仅有 $n=0$ 和 $n=1$。对于 $n\geqslant 2$,令 $k=\lfloor \lg n\rfloor$。恰恰有一项,其分母为 2^k,所以 $2^{k-1}H_n-\frac{1}{2}$是分母中仅含奇素数的诸项之和。如若 H_n 是一整数,则 $2^{k-1}H_n-\frac{1}{2}$将有等于 2 的分母。

20. 逐项展开被积函数。也见*AMM* **69** (1962), 239, 以及 H. W. Gould 的论文,*Mathematics Magazine* **34** (1961), 317 ~ 321。

21. $H_{n+1}^2-H_{n+1}^{(2)}$。

22. $(n+1)(H_n^2-H_n^{(2)})-2n(H_n-1)$。

23. $\Gamma'(n+1)/\Gamma(n+1)=1/n+\Gamma'(n)/\Gamma(n)$,因为 $\Gamma(x+1)=x\Gamma(x)$。因此 $H_n=\gamma+\Gamma'(n+1)/\Gamma(n+1)$。函数 $\psi(x)=\Gamma'(x)/\Gamma(x)=H_{x-1}-\gamma$ 称为 Ψ 函数或双伽玛函数。对于有理数 x 的某些值,见附录 A。

24. 它是 $x\lim\limits_{n\to\infty}e^{(H_n-\ln n)x}\prod\limits_{k=1}^{n}\left(\left(1+\frac{x}{k}\right)e^{-x/k}\right)=\lim\limits_{n\to\infty}\frac{x(x+1)\cdots(x+n)}{n^x n!}$。注:在上题中考虑的 H_n 之推广,因此等于 $H_x^{(r)}=\sum_{k\geqslant 0}\left(\frac{1}{(k+1)^r}-\frac{1}{(k+1+x)^r}\right)$,当 $r=1$ 时;对于更大的 r 值,亦可使用这同一思想。对所有复数 x,无穷乘积收敛。

1.2.8 小节

1. 在 k 个月之后有 F_{k+2}对,所以答案是 $F_{14}=377$ 对。

2. $\ln(\phi^{1000}/\sqrt{5})=1000\ \ln\phi-\frac{1}{2}\ln\ 5=480.40711$;$\log_{10}F_{1000}$是 $1/(\ln 10)$乘此数,或 208.64;因此 F_{1000}是一个 209 位的数,打头一位是 4。

4. 0,1,5;此后 F_n 增加非常快。

5. 0,1,12。

6. 用归纳法(这个等式对于负的 n 也成立;参照习题 8)。

7. 如果 d 是 n 的一个真因子,则 F_d 整除 F_n。假定 d 大于 2,则 F_d 大于 1 且小于 F_n。惟一的没有大于 2 的真因子的非素数是 $n=4$;$F_4=3$ 是仅有的例外。

8. $F_{-1}=1$;$F_{-2}=-1$;通过对 n 用归纳法,$F_{-n}=(-1)^{n+1}F_n$。

9. (15)不行。通过归纳论证,即假定某件事对于 n 和更大者为真,证明它对于 $n-1$ 为真,得证其它的都成立。

10. 当 n 是偶数时,它较大;当 n 是奇数时,它较小。(请见等式(14)。)

11. 归纳法;请见习题 9。这是习题 13(a)的特殊情形。

12. 如果 $\mathcal{G}(z)=\sum\mathcal{F}_n z^n$，则$(1-z-z^2)\mathcal{G}(z)=z+\mathcal{F}_0z^2+\mathcal{F}_1z^3+\cdots=z+z^2G(z)$。因此，$\mathcal{G}(z)=G(z)+zG(z)^2$；由等式(17)，我们求出 $\mathcal{F}_n=((3n+3)/5)F_n-(n/5)F_{n+1}$。

13. a) $a_n=rF_{n-1}+sF_n$。b) 因为$(b_{n+2}+c)=(b_{n+1}+c)+(b_n+c)$，我们可以考虑新的序列 $b'_n=b_n+c$。应用 a)于 b'_n，我们得到答案为 $cF_{n-1}+(c+1)F_n-c$。

14. $a_n=F_{m+n+1}+F_n-\binom{n}{m}-\binom{n+1}{m-1}-\cdots-\binom{n+m}{0}$。

15. $c_n=xa_n+yb_n+(1-x-y)F_n$。

16. F_{n+1}。用归纳法，以及$\binom{n+1-k}{k}=\binom{n-k}{k}+\binom{(n-1)-(k-1)}{k-1}$。

17. 一般地说，量$(x^{n+k}-y^{n+k})(x^{m-k}-y^{m-k})-(x^n-y^n)(x^m-y^m)$等于$(xy)^n(x^{m-n-k}-y^{m-n-k})(x^k-y^k)$。置 $x=\phi$，$y=\hat{\phi}$，并除以$(\sqrt{5})^2$。

18. 它是 F_{2n+1}。

19. 令 $u=\cos 72°$，$v=\cos 36°$。我们有 $u=2v^2-1$；$v=1-2\sin^2 18°=1-2u^2$。因此 $u+v=2(v^2-u^2)$，即，$1=2(v-u)=2v-4v^2+2$。我们得出结论 $v=\frac{1}{2}\phi$。（而且 $u=\frac{1}{2}\phi^{-1}$，$\sin 36°=\frac{1}{2}5^{1/4}\phi^{-1/2}$，$\sin 72°=\frac{1}{2}5^{1/4}\phi^{1/2}$。）

20. $F_{n+2}-1$。

21. 乘以 x^2+x-1；解答是$(x^{n+1}F_{n+1}+x^{n+2}F_n-x)/(x^2+x-1)$。如果分母为 0，则 x 为 $1/\phi$ 或 $1/\hat{\phi}$；于是解答是$(n+1-x^nF_{n+1})/(2x+1)$。

22. F_{m+2n}；在下一题中置 $t=2$。

23. $\frac{1}{\sqrt{5}}\sum_k\binom{n}{k}(\phi^kF_t^kF_{t-1}^{n-k}\phi^m-\hat{\phi}^kF_t^kF_{t-1}^{n-k}\hat{\phi}^m)=$

$$\frac{1}{\sqrt{5}}(\phi^m(\phi F_t+F_{t-1})^n-\hat{\phi}^m(\hat{\phi}F_t+F_{t-1})^n)=F_{m+tn}$$

24. F_{n+1}(第一行按余子式展开)。

25. $2^n\sqrt{5}F_n=(1+\sqrt{5})^n-(1-\sqrt{5})^n$。

26. 由费马定理，$2^{p-1}\equiv1$；现在应用上一题和习题 1.2.6-10(b)。

27. 如果 $p=2$，则命题为真。否则 $F_{p-1}F_{p+1}-F_p^2=-1$；因此由上题和费马定理，$F_{p-1}\cdot F_{p+1}\equiv0(\text{modulo } p)$。由于 $F_{p+1}=F_p+F_{p-1}$，因此在这些因子中仅仅有一个可为 p 的倍数。

28. $\hat{\phi}^n$。注：递推式 $a_{n+1}=Aa_n+B^n$，$a_0=0$ 的解为

如果 $A\neq B$，则 $a_n=(A^n-B^n)/(A-B)$；如果 $A=B$，则 $a_n=nA^{n-1}$

29. (a)

$\binom{n}{0}_\mathcal{F}$	$\binom{n}{1}_\mathcal{F}$	$\binom{n}{2}_\mathcal{F}$	$\binom{n}{3}_\mathcal{F}$	$\binom{n}{4}_\mathcal{F}$	$\binom{n}{5}_\mathcal{F}$	$\binom{n}{6}_\mathcal{F}$
1	0	0	0	0	0	0
1	1	0	0	0	0	0
1	2	2	1	0	0	0
1	3	6	3	1	0	0
1	5	15	15	5	1	0
1	8	40	60	40	8	1

(b) 由(6)推出。[É. Lucas, *Amer. J. Math.* **1** (1878), 201 ~ 204。]

30. 我们对 m 用归纳法进行论证，当 $m=1$ 时命题是显然的：

(a) $\sum_k \binom{m}{k}_{\mathscr{F}} (-1)^{\lceil (m-k)/2 \rceil} F_{n+k}^{m-2} F_k = F_m \sum_k \binom{m-1}{k-1}_{\mathscr{F}} (-1)^{\lceil (m-k)/2 \rceil} F_{n+k}^{m-2} = 0$。

(b) $\sum_k \binom{m}{k}_{\mathscr{F}} (-1)^{\lceil (m-k)/2 \rceil} F_{n+k}^{m-2} (-1)^k F_{m-k} =$

$$(-1)^m F_m \sum_k \binom{m-1}{k}_{\mathscr{F}} (-1)^{\lceil (m-1-k)/2 \rceil} F_{n+k}^{m-2} = 0。$$

(c) 由于 $(-1)^k F_{m-k} = F_{k-1}F_m - F_k F_{m-1}$ 且 $F_m \neq 0$，我们由(a)和(b)得出结论

$$\sum_k \binom{m}{k}_{\mathscr{F}} (-1)^{\lceil (m-k)/2 \rceil} F_{n+k}^{m-2} F_{k-1} = 0$$

(d) 由于 $F_{n+k} = F_{k-1}F_n + F_k F_{n+1}$，由(a)和(c)推出结果。这个结果也可以通过使用习题 1.2.6-58 中的"q 多项式定理"，以更一般的形式来进行证明。参考文献：Dov Jarden, *Recurring Sequences*, 第 2 版(Jerusalem, 1966), 30 ~ 33; J. Riordan, *Duke Math. J.* **29** (1962), 5 ~ 12。

31. 利用习题 8 和 11。

32. 取模 F_n，斐波那契序列为 $0, 1, \cdots, F_{n-1}, 0, F_{n-1}, -F_{n-2}, \cdots$。

33. 注意对于这个特定的 z，$\cos z = \frac{1}{2}(e^{iz} + e^{-iz}) = -i/2$；然后利用下列事实，即对于所有的 z，$\sin(n+1)z + \sin(n-1)z = 2\sin nz \cos z$。

34. 首先证明，F_{k_1} 的可能值是小于或等于 n 的最大斐波那契数；因而 $n - F_{k_1}$ 小于 $F_{k_1 - 1}$，而且由归纳法，有惟一的 $n - F_{k_1}$ 表示。这个证明的概要，十分类似于惟一因子分解定理之证明。斐波那契数系的讨论是由 E. Zeckendorf 给出的[见 *Simon Stevin* **29** (1952), 190 ~ 195; *Bull. Soc. Royale des Sciences de Liège* **41** (1972), 179 ~ 182]；习题 5.4.2-10 中讨论了推广。

35. 见 G. M. Bergman, *Mathematics Magazine* **31** (1957), 98 ~ 110。为表示 $x>0$，求使得 $\phi^k \leq x$ 的最大的 k 并且把 k 表示为 ϕ^k 加上 $x - \phi^k$ 的表示。

由 0 和 1 的平凡的表示开始，由以下的全整数递归规则出发，也可得到非负整数的表示：令 $L_n = \phi^n + \hat{\phi}^n = F_{n+1} + F_{n-1}$。对于 $0 \leq m \leq L_{2n-1}$ 和 $n \geq 1$，$L_{2n} + m$ 的表示是 $\phi^{2n} + \phi^{-2n-2}$ 加上 m 的表示。对于 $0 < m < L_{2n}$ 和 $n \geq 0$，$L_{2n+1} + m$ 的表示是 $\phi^{2n+1} + \phi^{-2n-2}$ 加上 $m - \phi^{2n}$ 的表示，其中后者是通过应用规则 $\phi^k - \phi^{k-2j} = \phi^{k-1} + \phi^{k-3} + \cdots + \phi^{k-2j+1}$ 得到的。结果是，所有的 0 和 1 的串 α，除以 $10^{2k}1$ 结尾的那些串之外，使得 α 以 1 开始而且没有相邻的 1，在恰好一个正整数的表示中出现在小数点的左边；后边那种串在这个表示中绝不出现。

36. 我们可以考虑无限的串 S_∞，因为对于 $n>1$ 的 S_n 由 S_∞ 的前 F_n 个字母组成。因此没有双重的 a，没有三重的 b。串 S_n 含有 F_{n-2} 个 a，F_{n-1} 个 b。如果我们像在习题 34 中那样，以斐波那契数系来表达 $m-1$，那么，S_∞ 的第 m 个字母是 a，当且仅当 $k_r = 2$。S_∞ 的第 k 个字母是 b，当且仅当 $\lfloor (k+1)\phi^{-1} \rfloor - \lfloor k\phi^{-1} \rfloor = 1$；在前 k 个字母中 b 的个数因此是 $\lfloor (k+1)\phi^{-1} \rfloor$。而且对于某个正整数 m 来说，当且仅当 $k = \lfloor m\phi \rfloor$ 时，第 k 个字母才是 b。John Bernoulli III 在 18 世纪，A. A. Markov 在 19 世纪以及随后许多其他数学家都研究了这个序列；请见 K. B. Stolarsky, *Canadian Math. Bull.* **19** (1976), 473 ~ 482。

37. [*Fibonacci Quart.* **1** (1963 年 12 月), 9 ~ 12。] 考虑习题 34 的斐波那契数系；如果在这个数

系中 $n=F_{k_1}+\cdots+F_{k_r}>0$，则令 $\mu(n)=F_{k_r}$。令 $\mu(0)=\infty$。我们发现：(A) 如果 $n>0$，则 $\mu(n-\mu(n))>2\mu(n)$。证明：$\mu(n-\mu(n))=F_{k_{r-1}}\geqslant F_{k_{r+2}}>2F_{k_r}$，因为 $k_r\geqslant 2$。(B)如果 $0<m<F_k$，则 $\mu(m)\leqslant 2(F_k-m)$。证明：令 $\mu(m)=F_j$；$m\leqslant F_{k-1}+F_{k-3}+\cdots+F_{j+(k-1-j)\bmod 2}=-F_{j-1+(k-1-j)\bmod 2}+F_k\leqslant-\frac{1}{2}F_j+F_k$。(C) 如果 $0<m<\mu(n)$，则 $\mu(n-\mu(n)+m)\leqslant 2(\mu(n)-m)$。证明：这可由(B)推出。(D)如果 $0<m<\mu(n)$，则 $\mu(n-m)\leqslant(n-m)\leqslant 2m$。证明：在(C)中置 $m=\mu(n)-m$。

现在我们证明，如果有 n 个筹码，而且如果在下轮中至多可取 q 个筹码，那么，有一种获胜的移动，当且仅当 $\mu(n)\leqslant q$。证明：(a) 如果 $\mu(n)>q$，则所有的移动都留下位置 n'，q'，且 $\mu(n')\leqslant q'$。[这可从上边的(D)推出。](b) 如果 $\mu(n)\leqslant q$，则我们或者能在这次移动中获胜(如果 $q\geqslant n$)，或者我们能进行留下位置 n'，q' 的一个移动，且 $\mu(n')>q'$。[这从上边的(A)推出：我们的移动是取 $\mu(n)$个筹码。]可以看出，假如 $j=1$ 或 $F_{k_{j-1}}>2(F_{k_j}+\cdots+F_{k_r})$，如果 $n=F_{k_1}+\cdots+F_{k_r}$，全部获胜的移动的集合，是对于某个满足 $1\leqslant j\leqslant r$ 的 j 取走 $F_{k_j}+\cdots+F_{k_r}$。

1000 的斐波那契表示是 987 + 13；惟一幸运的夺取胜利的移法，是移 13 个筹码。除非 n 是斐波那契数，否则第一个选手总能获胜。

更为一般得多的这种游戏的解答，已由 A. Schwenk 得到[*Fibonacci Quarterly* **8** (1970), 225 ~ 234]。

39. $(3^n-(-2)^n)/5$。

40. 对 m 用归纳法，我们证明，对于 $F_m<n<F_{m+1}$，$f(n)=m$：首先，$f(n)\leqslant\max(1+f(F_m),2+f(n-F_m))=m$。第二，如果 $f(n)<m$，则有某个 $k<n$，且使 $1+f(k)<m$(因此 $k\leqslant F_{m-1}$)和 $2+f(n-k)<m$(因此 $n-k\leqslant F_{m-2}$)；但就会有 $n\leqslant F_{m-1}+F_{m-2}$。[因此当右分支的开销是左分支的两倍时，在 6.2.1 小节中所定义的斐波那契树极小化从根到叶的极大开销。]

41. $F_{k_1+1}+\cdots+F_{k_r+1}=\phi n+(\hat\phi^{k_1}+\cdots+\hat\phi^{k_r})$是一个整数，且圆括号内的量介于 $\hat\phi^3+\hat\phi^5+\cdots=\phi^{-1}-1$ 和 $\hat\phi^2+\hat\phi^4+\cdots=\phi^{-1}$之间。类似地，$F_{k_1-1}+\cdots+F_{k_r-1}=\phi^{-1}n+(\hat\phi^{k_1}+\cdots+\hat\phi^{k_r})=f(\phi^{-1}n)$。[这样的斐波那契移动是用心算进行英里和公里之间转换的方便的方法；见*CMath*, §6.6。]

42. [*Fibonacci Quarterly* **6** (1968), 235 ~ 244。]如果有这种表示存在，对于所有整数 N，我们有

$$mF_{N-1}+nF_N=F_{k_1+N}+F_{k_2+N}+\cdots+F_{k_r+N} \qquad (*)$$

因此两个不同的表示将同习题 34 冲突。

反之，由归纳法我们可以证明对于所有非负整数 m 和 n 这种联合表示的存在性。但使用上一道题，并证明对于可能为负的整数 m 和 n 这样的联合表示存在当且仅当 $m+\phi n\geqslant 0$，是更为有趣的：令 N 足够地大，使得 $|m\hat\phi^{N-1}+n\hat\phi^N|<\phi^{-2}$，并且像在$(*)$中那样表示 $mF_{N-1}+nF_N$。于是 $mF_N+nF_{N+1}=\phi(mF_{N-1}+nF_N)+(m\hat\phi^{N-1}+n\hat\phi^N)=f(\phi(mF_{N-1}+nF_N))=F_{k_1+N+1}+\cdots+F_{k_r+N+1}$，由此得出，对于所有的 N，$(*)$成立。现在置 $N=0$ 和 $N=1$。

1.2.9 小节

1. $1/(1-2z)+1/(1-3z)$。

2. 由(6)得出,因为$\binom{n}{k} = n!/k!(n-k)!$。

3. $G'(z) = \ln(1/(1-z))/(1-z)^2 + 1/(1-z)^2$。由此以及 $G(z)/(1-z)$的意义,我们有$\sum_{k=1}^{n-1} H_k = nH_n - n$;这与等式 1.2.7-(8)一致。

4. 置 $t=0$。

5. 由(11)和(22),z^k 的系数为

$$\frac{(n-1)!}{k!}\sum_{0\le j<k}\left\{ {j \atop n-1} \right\}\binom{k}{j}$$

现在应用等式 1.2.6-(46)和 1.2.6-(52)。(或者,求微分并使用 1.2.6-(46)。)

6. $(\ln(1/(1-z)))^2$,导数是调和数之生成函数的两倍;因此,和为 $2H_{n-1}/n$。

8. $1/((1-z)(1-z^2)(1-z^3)\cdots)$。[这是历史上生成函数的最初的应用之一。有关欧拉在18世纪关于这个生成函数的有趣的研究,请看 G. Pólya, *Induction and Analogy in Mathematics* (Princeton: Princeton University Press, 1954),第 6 章。]

9. $\frac{1}{24}S_1^4 + \frac{1}{4}S_1^2S_2 + \frac{1}{8}S_2^2 + \frac{1}{3}S_3 + \frac{1}{4}S_4$。

10. $G(z) = (1+x_1z)\cdots(1+x_nz)$。如同推导等式(38)那样取对数,我们就有同样的公式,所不同的只是(24)代替(17),而且答案也完全相同,除了以 $-S_2, -S_4, -S_6, \cdots$来代替 $S_2, S_4, S_6, \cdots$之外。我们有 $a_1 = S_1$, $a_2 = \frac{1}{2}S_1^2 - \frac{1}{2}S_2$, $a_3 = \frac{1}{6}S_1^3 - \frac{1}{2}S_1S_2 + \frac{1}{3}S_3$, $a_4 = \frac{1}{24}S_1^4 - \frac{1}{4}S_1^2S_2 + \frac{1}{8}S_2^2 + \frac{1}{3}S_1S_3 - \frac{1}{4}S_4$。(参照习题 9。)类似于(39)的递推式是 $na_n = S_1a_{n-1} - S_2a_{n-2} + \cdots$。注:在这个递推式中的等式称为牛顿恒等式,因为它们首先出现在 Isaac Newton 的*Arithmetica Universalis* (1707)中;见 D. J. Struik, *Source Book in Mathematics* (Harvard University Press, 1969), 94 ~ 95。

11. 由于 $\sum_{m\ge1} S_mz^m/m = \ln G(z) = \sum_{k\ge1}(-1)^{k-1}(h_1z + h_2z^2 + \cdots)^k/k$,所求的系数是$(-1)^{k_1+k_2+\cdots+k_m-1}m(k_1+k_2+\cdots+k_m-1)!/k_1!\,k_2!\cdots k_m!$。[乘以$(-1)^{m-1}$得到当借助于习题10的诸 a 表达 S_m 时 $a_1^{k_1}a_2^{k_2}\cdots a_m^{k_m}$ 的系数。Albert Girard 在他的*Invention Nouvelle en Algébre* (Amsterdam: 1629)一书接近结尾处列出了借助 a_1, a_2, a_3 及 a_4 表达 S_1, S_2, S_3 和 S_4 的公式;由此诞生了对称函数理论。]

12. $\sum_{m,n\ge0} a_{mn}w^mz^n = \sum_{m,n\ge0}\binom{n}{m}w^mz^n = \sum_{n\ge0}(1+w)^nz^n = 1/(1-z-wz)$。

13. $\int_n^{n+1} e^{-st}f(t)\,dt = (a_0+\cdots+a_n)(e^{-sn} - e^{-s(n+1)})/s$。对所有的 n,把这些表达式加在一起,我们求得 $\mathbf{L}f(s) = G(e^{-s})/s$。

14. 参照习题 1.2.6-38。

15. $G_n(z) = G_{n-1}(z) + zG_{n-2}(z) + \delta_{n0}$,所以我们求得 $H(w) = 1/(1-w-zw^2)$。因此,最终地,我们求得当 $z \ne -\frac{1}{4}$时,

$$G_n(z) = \left(\left(\frac{1+\sqrt{1+4z}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{1+4z}}{2}\right)^{n+1}\right)\Big/\sqrt{1+4z}$$

于对 $n\ge0$, $G_n(-1/4) = (n+1)/2^n$。

16. $G_{nr}(z) = (1+z+\cdots+z^r)^n = \left(\frac{1-z^{r+1}}{1-z}\right)^n$。[注意 $r=\infty$的情况。]

17. $\sum_k \binom{-w}{k}(-z)^k = \sum_k \frac{w(w+1)\cdots(w+k-1)}{k(k-1)\cdots 1} z^k = \sum_{n,k} \left[{k \atop n} \right] z^k w^n / k!$。(或者,把它写成 $e^{w\ln(1/(1-z))}$而且首先按 w 的幂展开。)

18. (a)对于固定的 n 和变动的 r,由等式(27),生成函数是

$$G_n(z) = (1+z)(1+2z)\cdots(1+nz) = z^{n+1}\left(\frac{1}{z}\right)\left(\frac{1}{z}+1\right)\left(\frac{1}{z}+2\right)\cdots\left(\frac{1}{z}+n\right) = \sum_k \left[{n+1 \atop k} \right] z^{n+1-k}$$

因此,答案为$\left[{n+1 \atop n+1-r} \right]$。(b)类似地,由等式(28),生成函数是

$$\frac{1}{1-z}\cdot\frac{1}{1-2z}\cdot\cdots\cdot\frac{1}{1-nz} = \sum_k \left\{ {k \atop n} \right\} z^{k-n}$$

所以答案为$\left\{ {n+r \atop n} \right\}$。

19. $\sum_{n\geq 1}(1/n - 1/(n+p/q))x^{p+nq} = \sum_{k=0}^{q-1}\omega^{-kp}\ln(1-\omega^k x) - x^p\ln(1-x^q) + \frac{q}{p}x^p = f(x) + g(x)$,其中 $\omega = e^{2\pi i/q}$,而且 $f(x) = \sum_{k=1}^{q-1}\omega^{-kp}\ln(1-\omega^k x)$,$g(x) = (1-x^p)\ln(1-x) + \frac{q}{p}x^p - x^p\ln\frac{1-x^q}{1-x}$。

现在 $\lim_{x\to 1-} g(x) = q/p - \ln q$。由恒等式

$$\ln(1-e^{i\theta}) = \ln\left(2e^{i(\theta-\pi)/2}\,\frac{e^{i\theta/2}-e^{-i\theta/2}}{2i}\right) = \ln 2 + \frac{1}{2}i(\theta-\pi) + \ln\sin\frac{\theta}{2}$$

我们可以写 $\lim_{x\to 1-} f(x) = f(1) = A + B$,其中

$$A = \sum_{k=1}^{q-1}\omega^{-kp}\left(\ln 2 - \frac{i\pi}{2} + \frac{ik\pi}{q}\right) = -\ln 2 + \frac{i\pi}{2} + \frac{i\pi}{(\omega^{-p}-1)}$$

$$B = \sum_{k=1}^{q-1}\omega^{-kp}\ln\sin\frac{k}{q}\pi = \sum_{0<k<q/2}(w^{-kp} + w^{-(q-k)p})\ln\sin\frac{k}{q}\pi = 2\sum_{0<k<q/2}\cos\frac{2pk}{q}\pi\cdot\ln\sin\frac{k}{q}\pi$$

最后,

$$\frac{i}{2} + \frac{i}{(\omega^{-p}-1)} = -\frac{i}{2}\left(\frac{1+\omega^p}{1-\omega^p}\right) = -\frac{i}{2}\left(\frac{\omega^{p/2}+\omega^{-p/2}}{\omega^{p/2}-\omega^{-p/2}}\right) = \frac{1}{2}\cot\frac{p}{q}\pi$$

[高斯在他关于超几何级数的专著的 § 33 中推导了这些结果,即等式[75],但未作充分证明;阿贝尔在*Crelle* **1** (1826),314~315 提供了一种证明。]

20. 由等式 1.2.6-(45),$c_{mk} = k!\left\{ {m \atop k} \right\}$。

21. 我们求得 $z^2G'(z)+zG(z)=G(z)-1$,这个微分方程的解是 $G(z)=(-1/z)e^{-1/z}\cdot(E_1(-1/z)+C)$,其中 $E_1(z)=\int_z^\infty e^{-t}dt/t$, C 是常数。在 $z=0$ 的邻域中这个函数是奇异的,因此 $G(z)$没有幂级数展开。确实,由于$\sqrt[n]{n!}\approx n/e$ 无界,在此情况下生成函数不收敛;然而,当 $z<0$ 时,它是对于所述函数的渐近展开。[请见 K. Knopp, *Infinite Sequences and Series* (Dover, 1956),第 66 节]

22. $G(z)=(1+z)^r(1+z^2)^r(1+z^4)^r(1+z^8)^r\cdots=(1-z)^{-r}$。由此得出所述的和是 $\binom{r+n-1}{n}$。

23. (a)当 $m=1$ 时,这是二项式定理,同时 $f_1(z)=z$ 且 $g_1(z)=1+z$。当 $m\geqslant 1$ 时,如果我们以 $z_m(1+z_{m+1}^{-1})$代替z_m,并令 $f_{m+1}(z_1,\cdots,z_{m+1})=z_{m+1}f_m(z_1,\cdots,z_{m-1},z_m(1+z_{m+1}^{-1}))$, $g_{m+1}(z_1,\cdots,z_{m+1})=z_{m+1}g_m(z_1,\cdots,z_{m-1},z_m(1+z_{m+1}^{-1}))$,可使 m 增 1。因此 $g_2(z_1,z_2)=z_1+z_2+z_1z_2$,且

$$\frac{g_m(z_1,\cdots,z_m)}{f_m(z_1,\cdots,z_m)}=1+\cfrac{z_1^{-1}}{1+\cfrac{z_2^{-1}}{1+\cfrac{\ddots}{1+z_m^{-1}}}}$$

f_m 和 g_m 两个多项式都满足相同的递推式$f_m=z_mf_{m-1}+z_{m-1}f_{m-2}$, $g_m=z_mg_{m-1}+z_{m-1}g_{m-2}$,以及初始条件 $f_{-1}=0$, $f_0=g_{-1}=g_0=z_0=1$。由此得出,g_m 是由 $z_1\cdots z_m$ 开始并且删去零个或更多个不相邻因子的可得到的所有项之和;有 F_{m+2}个方法这样做。除了 z_1 必须保留外,类似的解释适用于 f_m。在(b)部分中,我们将遇到多项式 $h_m=z_mg_{m-1}+z_{m-1}f_{m-2}$;这是由 $z_1\cdots z_m$ 开始通过去掉不是循环地相邻的因子可得到的所有项之和。例如,$h_3=z_1z_2z_3+z_1z_2+z_1z_3+z_2z_3$。

(b) 由(a)部分,$S_n(z_1,\cdots,z_{m-1},z)=[z_m^n]\sum_{r=0}^n z^r z_m^{n-r}f_m^{n-r}g_m^r$;因此

$$S_n(z_1,\cdots,z_m)=\sum_{0\leqslant s\leqslant r\leqslant n}\binom{r}{s}\binom{n-r}{s}a^{r-s}b^sc^sd^{n-r-s}$$

其中,$a=z_mg_{m-1}$, $b=z_{m-1}g_{m-2}$, $c=z_mf_{m-1}$, $d=z_{m-1}f_{m-2}$。以 z^n 来乘这个等式,并且首先对 n,然后对 r,然后对 s 求和,得到封闭形式

$$S_n(z_1,\cdots,z_m)=[z^n]\frac{1}{(1-az)(1-dz)-bcz^2}$$

$$=\frac{\rho^{n+1}-\sigma^{n+1}}{\rho-\sigma}$$

其中 $1-(a+d)z+(ad-bc)z^2=(1-\rho z)(1-\sigma z)$。这里 $a+d=h_m$, $ad-bc$ 简化成$(-1)^mz_1\cdots z_m$。[碰巧我们建立了递推式 $S_n=h_mS_{n-1}-(-1)^mz_1\cdots z_mS_{n-2}$,如果没有生成函数的帮助这是不容易推导的关系。]

(c)令 $\rho_1=(z+\sqrt{z^2+4z})/2$ 和 $\sigma_1=(z-\sqrt{z^2+4z})/2$ 是当 $m=1$ 时的根;于是 $\rho_m=\rho_1^m$ 和 $\sigma_m=\sigma_1^m$。

Carlitz 利用这个结果推导出令人惊讶的事实:"右对齐的二项式系数"的 $n\times n$ 矩阵

$$A=\begin{pmatrix} 0 & 0 & \cdots & 0 & \binom{0}{0} \\ 0 & 0 & \cdots & \binom{1}{0} & \binom{1}{1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \binom{n-1}{0} & \binom{n-1}{1} & \cdots & \binom{n-1}{n-2} & \binom{n-1}{n-1} \end{pmatrix}$$

的特征多项式 $\det(xI-A)$是$\sum_k \binom{n}{k}_{\mathscr{F}}(-1)^{\lceil (n-k)/2\rceil}x^k$,且有斐波那契系数(见习题 1.2.8-30)。利用类似的方法,他还证明

$$\sum_{k_1,\cdots,k_m\geqslant 0}\binom{k_1+k_2}{k_1}\binom{k_2+k_3}{k_2}\cdots\binom{k_m+k_1}{k_m}z_1^{k_1}\cdots z_m^{k_m}=\frac{1}{\sqrt{z_1^2\cdots z_m^2h_m(-z_1^{-1},\cdots,-z_m^{-1})^2-4z_1\cdots z_m}}$$

[*Collectanea Math.* **27** (1965),281 ~ 296。]

24. 两边都等于 $\sum_k\binom{m}{k}[z^n](zG(z))^k$。当 $G(z)=1/(1-z)$ 时,恒等式变成 $\sum_k\binom{m}{k}\binom{n-1}{n-k}=\binom{m+n-1}{n}$,这是 1.2.6-(21)中的一个情况。当 $G(z)=(e^z-1)/z$ 时,它变成$\sum_k m^k\left\{ {n \atop k} \right\}=m^n$,即等式 1.2.6-(45)。

25. $\sum_k[w^k](1-2w)[z^n]z^k(1+z)^{2n-2k}=[z^n](1+z)^{2n}\sum_k[w^k](1-2w)^n(z/(1+z)^2)^k$,它等于$[z^n](1+z)^{2n}/(1-2z/(1+z)^2)^n=[z^n](1+z^2)^n=\binom{n}{n/2}[n\text{ 偶}]$。类似地我们求得 $\sum_k\binom{n}{k}\binom{2n-2k}{n-k}(-4)^k=(-1)^n\binom{2n}{n}$。这种求和技术的许多例子都可在 G. P. Egorychev 的 *Integral Representation and the Computation of Combinatorial Sums* 一书,译自 1977 年的俄罗斯版本(Amer. Math. Soc.,1984)。

26. $[F(z)]G(z)$表示 $F(z^{-1})G(z)$的常数项。请见 D. E. Knuth 在*A Classical Mind* (Prentice-Hall,1994),247 ~ 258 中的讨论。

1.2.10 小节

1. $G_n(0)=1/n$;这是 $X[n]$为最大的概率。
2. $G''(1)=\sum_k k(k-1)p_k$, $G'(1)=\sum_k kp_k$。
3. (最小值为 0,平均值为 6.49,最大值为 999,标准差 2.42。)注意 $H_n^{(2)}$ 近似于 $\pi^2/6$;见等式 1.2.7-(7)。
4. $\binom{n}{k}p^kq^{n-k}$。
5. 平均值是 $36/5=7.2$;标准差是 $6\sqrt{2}/5\approx1.697$。

6. 对于(18),公式

$$\ln(q+pe^t) = \ln\left(1+pt+\frac{pt^2}{2}+\frac{pt^3}{6}+\cdots\right) =$$

$$pt+p(1-p)\frac{t^2}{2}+p(1-p)(1-2p)\frac{t^3}{6}+\cdots$$

告诉我们 $\kappa_3/n = p(1-p)(1-2p) = pq(q-p)$。(这个很好的模式不继续到 t^4 的系数上。)在分布(8)的情况下,设置 $p=k^{-1}$给了我们 $\kappa_3 = \sum_{k=2}^{n} k^{-1}(1-k^{-1})(1-2k^{-1}) = H_n - 3H_n^{(2)} + 2H_n^{(3)}$。而对于(20),我们有 $\ln G(e^t) = t + H(nt) - H(t)$,其中 $H(t) = \ln((e^t-1)/t)$。由于 $H'(t) = e^t/(e^t-1) - 1/t$,在这种情况下对于所有 $r \geqslant 2$,我们有 $\kappa_r = (n^r - 1)B_r/r$;特别是 $\kappa_3 = 0$。

7. $A=k$ 的概率是 p_{mk}。因为我们可以把这些值看成 $1,2,\cdots,m$。给定任何把 n 个位置分划成 m 个不相交集合的分划,有 $m!$ 种方法来分配数 $1,\cdots,m$ 到这些集合中。算法 M 把这些值处理成就像是仅仅每个集合的最右元素存在一样;所以,p_{mk}是对于任何固定的分划的平均值。例如,如果 $n=5, m=3$,一种分划是

$$\{X[1],X[4]\} \qquad \{X[2],X[5]\} \quad \{X[3]\}$$

可能的排列是 12312,13213,21321,23123,31231,32132。在每个分划中,我们以 $A=k$ 得到相同百分数的排列。

另一方面,如果给出更多的信息,则概率分布即改变。如果 $n=3, m=2$,则上段的论证考虑六种可能性 122,212,221,211,121,112;如果我们知道有两个 2 和一个 1,则仅仅要考虑这些可能性中的前三种。但是,这个解释与本习题的命题已不一致了。

8. $M^{\underline{n}}/M^n$。M 越大,则概率就越接近 1。

9. 命 q_{nm}是恰有 m 个不同的值出现的概率;则从递推式

$$q_{nm} = \frac{M-m+1}{M} q_{(n-1)(m-1)} + \frac{m}{M} q_{(n-1)m}$$

我们导出

$$q_{nm} = M! \left\{ {n \atop m} \right\} \Big/ (M-m)!\, M^n$$

也见习题 1.2.6-64。

10. 这是对所有 m 求和的 $q_{nm}p_{mk}$;即

$$\frac{1}{M^n} \sum_m \binom{M}{m} \left\{ {n \atop m} \right\} \left[{m \atop k+1} \right]$$

这好像并不是关于平均值的简单公式,它比

$$H_M - \sum_{m=1}^{M} \left(1 - \frac{m}{M}\right)^n m^{-1} = H_n + \sum_{k=1}^{n} \left(\binom{n}{k} - 1 \right) B_k M^{-k} k^{-1}$$

小 1。

11. 由于这是乘积,我们加上每项的半不变量。如果 $H(z) = z^n$,则 $H(e^t) = e^{nt}$,所以我们发现 $\kappa_1 = n$ 且所有其它的为 0。因此,mean$(F) = n +$ mean(G),而且所有其它的半不变量均不变(这是“半不变量”名称之由来)。

12. 通过写出 e^{kt} 的幂级数，第一个恒等式是显然的。对于第二个，令 $u = 1 + M_1t + M_2t^2/2! + \cdots$；当 $t = 0$ 时，我们有 $u = 1$ 和 $D_t^ku = M_k$。而且 $D_u^j(\ln u) = (-1)^{j-1}(j-1)!/u^j$。由习题 11，除了我们留下有 $k_1 > 0$ 的所有项之外，相同的公式可应用于中心的矩量；因此，$\kappa_2 = m_2$，$\kappa_3 = m_3$，$\kappa_4 = m_4 - 3m_2^2$。

13. $G_n(z) = \dfrac{\Gamma(n+z)}{\Gamma(z+1)n!} = \dfrac{e^{-z}(n+z)^{z-1}}{\Gamma(z+1)}\left(1+\dfrac{z}{n}\right)^n(1+O(n^{-1})) = \dfrac{n^{z-1}}{\Gamma(z+1)}(1+O(n^{-1}))$。

令 $z_n = e^{it/\sigma_n}$。当 $n \to \infty$ 和 t 被固定时，我们有 $z_n \to 1$；因此 $\Gamma(z_n+1) \to 1$，且

$$\lim_{n\to\infty} z_n^{-\mu_n}G_n(z_n) = \lim_{n\to\infty}\exp\left(\frac{-it\mu_n}{\sigma_n} + (e^{it/\sigma_n} - 1)\ln n\right) =$$

$$\lim_{n\to\infty}\exp\left(\frac{-t^2\ \ln n}{2\sigma_n^2} + O\left(\frac{1}{\sqrt{\log n}}\right)\right) = e^{-t^2/2}$$

注：这是 Goncharov 定理[*Izv. Akad. Nauk SSSR Ser. Math.* **8** (1944), 3 ~ 48]。P. Flajolet 和 M. Soria [*Disc. Math* . **114** (1993), 159 ~ 180]已经推广这个分析以证明 $G_n(z)$ 和相关分布的很大的族不但在接近平均值时是近似地正态的，而且对于某个正常数 a 和对于所有的 n 和 x，在

$$\text{概率}\left(\left|\frac{X_n - \mu_n}{\sigma_n}\right| > x\right) < e^{-ax}$$

的意义下，它们还有一致的指数的尾部。

14. $e^{-itpn/\sqrt{pqn}}(q + pe^{it/\sqrt{pqn}})^n = (qe^{-itp/\sqrt{pqn}} + pe^{itq/\sqrt{pqn}})^n$。把指数展开成幂级数，得到$(1 - t^2/2n + O(n^{-3/2}))^n = \exp(n\ \ln(1 - t^2/2n + O(n^{-3/2}))) = \exp(-t^2/2 + O(n^{-1/2})) \to \exp(-t^2/2)$。

15. a) $\sum_{k\geq 0}e^{-\mu}(\mu z)^k/k! = e^{\mu(z-1)}$。b) $\ln e^{\mu(e^t-1)} = \mu(e^t - 1)$，所以所有半不变量均等于 μ。c) $\exp(-itnp/\sqrt{np})\exp(np(it/\sqrt{np} - t^2/2np + O(n^{-3/2}))) = \exp(-t^2/2 + O(n^{-1/2}))$。

16. $g(z) = \sum_k p_k\, g_k(z)$；$\text{mean}(g) = \sum_k p_k\ \text{mean}(g_k)$；且 $\text{var}(g) = \sum_k p_k\ \text{var}(g_k) + \sum_{j<k} p_j p_k(\text{mean}(g_j) - \text{mean}(g_k))^2$。

17. a) $f(z)$和 $g(z)$的系数非负而且 $f(1) = g(1) = 1$。显然，$h(z)$享有同样的特征，因为 $h(1) = g(f(1))$，而且 h 的系数是具有非负系数的f, g 之系数的多项式。(b)令 $f(z) = \sum p_kz^k$，其中 p_k 是某个事件得“分”为 k 的概率。令 $g(z) = \sum q_kz^k$，其中 q_k 是由f 描述的事件恰恰发生 k 次的概率(事件的每一出现是独立于其它事件的)。则 $h(z) = \sum r_kz^k$，其中 r_k 是出现的事件的得分之和等于k 的概率。(这是容易看出的，如果我们注意到 $f(z)^k = \sum s_tz^t$ 的话，其中 s_t 是在事件的 k 次独立出现中得到总分 t 的概率。)例子：如果 f 给出一个男人有 k 个男孩的概率，而且如果 g 给出在第 n 代有 k 个男子的概率，则 h 就给出在第 $n+1$ 代有 k 个男子的概率——假定无关性。c) $\text{mean}(h) = \text{mean}(g)\text{mean}(f)$；$\text{var}(h) = \text{var}(g)\text{mean}^2(f) + \text{mean}(g)\text{var}(f)$。

18. 把 $X[1], \cdots, X[n]$的选择考虑作一个过程，其中我们首先放置所有的 n，然后在这些 n 当中放置所有的$n-1$，…，最后在剩余的当中放置 1。当我们在数 $r+1, \cdots, n$ 当中放置诸 r 时，从右到左的局部极大值的个数增加 1，当且仅当在最右端放置一个 r。这以 $k_r/(k_r + k_{r+1} + \cdots + k_n)$的概率出现。

19. 令 $a_k = l$。于是 a_k 是 $a_1\cdots a_n$ 自左到右的极大值$\Leftrightarrow j < k$ 意味着 $a_j < l \Leftrightarrow a_j > l$ 意味着 $j > k \Leftrightarrow j > l$ 意味着 $b_j > k \Leftrightarrow k$ 是 $b_1\cdots b_n$ 的自右到左的极小值。

20. 我们有 $m_L=\max\{a_1-b_1,\cdots,a_n-b_n\}$。证明：如果不然，令 k 是使得 $a_k-b_k>m_L$ 的最小下标。则 a_k 不是自左到右的极大值，所以有 $j<k$ 且 $a_j\geqslant a_k$。但由此得出 $a_j-b_j\geqslant a_k-b_k>m_L$，同 k 是极小值矛盾。类似地，$m_R=\max\{b_1-a_1,\cdots,b_n-a_n\}$。

21. 当 $\epsilon\geqslant q$ 时这个结果是平凡的，所以我们可以假定 $\epsilon<q$。在(25)中置 $x=\frac{p+\epsilon}{p}\frac{q}{q-\epsilon}$ 给出 $\Pr(X\geqslant n(p+\epsilon))\leqslant\left(\left(\frac{p}{p+\epsilon}\right)^{p+\epsilon}\left(\frac{q}{q-\epsilon}\right)^{q-\epsilon}\right)^n$。现在 $\left(\frac{p}{p+\epsilon}\right)^{p+\epsilon}\leqslant e^{-\epsilon}$，因为对于所有实数 t，$t\leqslant e^{t-1}$。而且 $(q-\epsilon)\ln\frac{q}{q-\epsilon}=\epsilon-\frac{1}{2\cdot1}\epsilon^2q^{-1}\frac{1}{3\cdot2}\epsilon^3q^{-2}-\cdots\leqslant\epsilon-\frac{1}{2q}\epsilon^2$。(当 $p\geqslant\frac{1}{2}$ 时，更详细的分析产生出稍强的估计 $\exp(-\epsilon^2n/(2pq))$；更进一步的工作产生对于所有 p 的上界 $\exp(-2\epsilon^2n)$。)

通过调换头部和尾部的作用，我们求得

$$\Pr(X\leqslant n(p-\epsilon))=\Pr(n-X\geqslant n(q+\epsilon))\leqslant e^{-\epsilon^2n/(2p)}$$

(不应把这里的"尾部"同概率分布的尾部相混淆。)

22. (a) 在(24)和(25)中置 $x=r$，而且注意 $q_k+p_kr=1+(r-1)p_k\leqslant e^{(r-1)p_k}$。[见 H. Chernoff, *Annals of Math. Stat.* **23** (1952), 493～507。]

(b) 令 $r=1+\delta$，其中 $|\delta|\leqslant1$。于是 $r^{-r}e^{r-1}=\exp\left(-\frac{1}{2\cdot1}\delta^2+\frac{1}{3\cdot2}\delta^3-\cdots\right)$，当 $\delta\leqslant0$ 时 $r^{-r}e^{r-1}\leqslant e^{\delta^2/2}$，当 $\delta\geqslant0$ 时 $r^{-r}e^{r-1}\leqslant e^{-\delta^2/3}$。

(c) 随着 r 从 1 增到 ∞ 时，函数 $r^{-1}e^{1-r^{-1}}$ 从 1 减为 0。如果 $r\geqslant2$，函数值 $\leqslant\frac{1}{2}e^{1/2}<.825$；如果 $r\geqslant4.32$，函数值 $<\frac{1}{2}$。

顺便说一句，当 X 有习题 15 的泊松分布时，$x=r$ 时的尾部不等式精确地给出相同的估计 $(r^{-r}e^{r-1})\mu$。

23. 在(24)中置 $x=\frac{p-\epsilon}{p}\frac{q}{q-\epsilon}$ 给出 $\Pr(X\leqslant n(p-\epsilon))\leqslant\left(\left(\frac{p}{p-\epsilon}\right)^{p-\epsilon}\left(\frac{q-\epsilon}{q}\right)^{q-\epsilon}\right)^n\leqslant e^{-\epsilon^2n/(2qp)}$。类似地，$x=\frac{p+\epsilon}{p}\frac{q}{q+\epsilon}$ 产生出 $\Pr(X\geqslant n(p+\epsilon))\leqslant\left(\left(\frac{p}{p+\epsilon}\right)^{p+\epsilon}\left(\frac{q+\epsilon}{q}\right)^{q+\epsilon}\right)^n$。令 $f(\epsilon)=(q+\epsilon)\ln\left(1+\frac{\epsilon}{q}\right)-(p+\epsilon)\ln\left(1+\frac{\epsilon}{p}\right)$，并且注意 $f'(\epsilon)=\ln\left(1+\frac{\epsilon}{q}\right)-\ln\left(1+\frac{\epsilon}{p}\right)$。由此得出，如果 $0\leqslant\epsilon\leqslant p$，$f(\epsilon)\leqslant-\epsilon^2/(6pq)$。

1.2.11.1 小节

1. 0。

2. 每个 O 符号表示不同的近似量；因为左边可以是 $f(n)-(-f(n))=2f(n)$，我们充其量只能说是 $O(f(n))-O(f(n))=O(f(n))$，这可从(6)和(7)式推出。为证明(7)式，注意到，如果当 $n\geqslant n_0$ 时 $|x_n|\leqslant M|f(n)|$，而当 $n\geqslant n'_0$ 时 $|x'_n|\leqslant M'|f(n)|$，则当 $n\geqslant\max(n_0,n'_0)$ 时，$|x_n\pm x'_n|\leqslant|x_n|+|x'_n|\leqslant(M+M')|f(n)|$。(学生 J. H. Quick 得出。)

3. $n(\ln n)+\gamma n+O(\sqrt{n}\ln n)$。

4. $\ln a+(\ln a)^2/2n+(\ln a)^3/6n^2+O(n^{-3})$。

5. 如果 $f(n)=n^2$ 和 $g(n)=1$，则 n 属于集合 $O(f(n)+g(n))$ 但不属于集合 $f(n)+O(g(n))$。所以该命题不成立。

6. O 符号的可变的个数 n,已为单个的 O 符号所代替,错误地蕴涵着一单个的 M 值对于每个项 $|kn| \leqslant Mn$ 都将满足。如我们所知道的,给定的和式实际上是 $\Theta(n^3)$。最后的等式 $\sum_{k=1}^{n} O(n) = O(n^2)$,完全正确。

7. 如果 x 是正数,则幂级数 1.2.9-(22)告诉我们 $e^x > x^{m+1}/(m+1)!$;因此 e^x/x^m 的比率不能为任何 M 所限定。

8. 以 e^n 代替 n,而且应用前面习题的方法。

9. 如果对于 $|z| \leqslant r$ 有 $|f(z)| \leqslant M|z|^m$,则 $e^{f(z)} \leqslant e^{M|z|^m} = 1 + |z|^m(M + M^2|z|^m/2! + M^3|z|^{2m}/3! + \cdots) \leqslant 1 + |z|^m(M + M^2r^m/2! + M^3r^{2m}/3! + \cdots)$。

10. 如果 m 是正整数,则 $\ln(1 + O(z^m)) = O(z^m)$。证明:如果 $f(z) = O(z^m)$,则存在正数 $r < 1$,$r' < 1$ 和常数 M,使得当 $|z| \leqslant r$ 时,$|f(z)| \leqslant M|z|^m \leqslant r'$。于是,$|\ln(1+f(z))| \leqslant |f(z)| + \frac{1}{2}|f(z)|^2 + \cdots \leqslant |z|^m M(1 + \frac{1}{2}r' + \cdots)$。

11. 我们能以 $m=1$ 和 $z = \ln n / n$ 来应用等式(12)。这是正当的,因为对于任何给定的 $r > 0$,当 n 充分大时,$\ln n / n \leqslant r$。

12. 令 $f(z) = (ze^z/(e^z - 1))^{1/2}$。若 $\begin{bmatrix} 1/2 \\ 1/2 - k \end{bmatrix}$ 是 $O(n^k)$,则所述的恒等式将表明 $[z^k]f(z) = O(n^k/(k-1)!)$,所以当 $z = 2\pi i$ 时 $f(z)$ 将收敛。但是 $f(2\pi i) = \infty$。

13. 证明:在 O 和 Ω 的定义下我们可以取 $L = 1/M$。

1.2.11.2 小节

1. $(B_0 + B_1z + B_2z^2/2! + \cdots)e^z = (B_0 + B_1z + B_2z^2/2! + \cdots) + z$;应用等式 1.2.9-(11)。

2. 为进行分部积分,函数 $B_{m+1}(\{x\})$ 必须是连续的。

3. $|R_{mn}| \leqslant |B_m/(m)!| \int_1^n |f^{(m)}(x)|\,dx$。

[注:我们有 $B_m(x) = (-1)^m B_m(1-x)$,而且 $B_m(x)$ 是 $m!$ 乘以在 $ze^{xz}/(e^z-1)$ 中 z^m 的系数。特别是,由于 $e^{z/2}/(e^z-1) = 1/(e^{z/2}-1) - 1/(e^z-1)$,我们有 $B_m\left(\frac{1}{2}\right) = (2^{1-m}-1)B_m$。不难证明,当 m 为偶数,$0 \leqslant x \leqslant 1$ 时,$|B_m - B_m(x)|$ 的最大值出现在 $x = \frac{1}{2}$ 处。现在当 $m = 2k \geqslant 4$ 时,让我们简单地写 R_n 和 C_m 代表 R_{mn} 和 C_{mn}。我们有 $R_{m-2} = C_m + R_m = \int_1^n (B_m - B_m(\{x\}))f^{(m)}(x)\,dx/m!$,而且 $B_m - B_m(\{x\})$ 介于 0 和 $(2-2^{1-m})B_m$ 之间;因此 R_{m-2} 介于 0 和 $(2-2^{1-m})C_m$ 之间。由此可推出 R_m 处于 $-C_m$ 和 $(1-2^{1-m})C_m$ 之间,这是个略强的结果。按照这一论证,我们看到,对于 $1 < x < n$,如果 $f^{(m+2)}(x)f^{(m+4)}(x) > 0$,则 C_{m+2} 和 C_{m+4} 具有相反的符号,R_m 和 C_{m+2} 符号相同,R_{m+2} 和 C_{m+4} 符号相同,且 $|R_{m+2}| \leqslant |C_{m+2}|$;这就证明了(13)。请见 J. F. Steffensen, *Interpolation* (Baltimore: 1937), § 14。]

4. $\sum_{0 \leqslant k < n} k^m = \frac{n^{m+1}}{1+m} + \sum_{k=1}^{m} \frac{B_k}{k!}\frac{m!}{(m-k+1)!}n^{m-k+1} = \frac{1}{m+1}B_{m+1}(n) - \frac{1}{m+1}B_{m+1}$。

5. 由此得出

$$\kappa = \sqrt{2}\lim_{n\to\infty}\frac{2^{2n}(n!)^2}{\sqrt{n}(2n)!}$$

$$\kappa^2 = \lim_{n\to\infty}\frac{2}{n}\frac{n^2(n-1)^2\cdots(1)^2}{\left(n-\frac{1}{2}\right)^2\left(n-\frac{3}{2}\right)^2\cdots\left(\frac{1}{2}\right)^2} = 4\,\frac{2\cdot2\cdot4\cdot4\cdots}{1\cdot3\cdot3\cdot5\cdots} = 2\pi$$

6. 假定 $c>0$ 并考虑 $\sum_{0\le k<n}\ln(k+c)$。我们得出

$$\ln(c(c+1)\cdots(c+n-1)) = (n+c)\ln(n+c) - c\ln c - n - \frac{1}{2}\ln(n+c) + \frac{1}{2}\ln c + \sum_{1<k\le m}\frac{B_k(-1)^k}{k(k-1)}\left(\frac{1}{(n+c)^{k-1}} - \frac{1}{c^{k-1}}\right) + R_{mn}$$

而且

$$\ln(n-1)! = \left(n-\frac{1}{2}\right)\ln n - n + \sigma + \sum_{1<k\le m}\frac{B_k(-1)^k}{k(k-1)}\left(\frac{1}{n^{k-1}}\right) - \frac{1}{m}\int_n^\infty \frac{B_m(\{x\})dx}{x^m}$$

现在 $\ln\Gamma_{n-1}(c) = c\ln(n-1) + \ln(n-1)! - \ln(c\cdots(c+n-1))$；代入并令 $n\to\infty$，我们得到

$$\ln\Gamma(c) = -c + \left(c-\frac{1}{2}\right)\ln c + \sigma + \sum_{1<k\le m}\frac{B_k(-1)^k}{k(k-1)c^{k-1}} - \frac{1}{m}\int_0^\infty \frac{B_m(\{x\})dx}{(x+c)^m}$$

这证明了 $\Gamma(c+1) = ce^{\ln\Gamma(c)}$ 有着等同于我们对 $c!$ 导出的展开式。

7. $An^{n^2/2+n/2+1/12}e^{-n^2/4}$，其中 A 是一常数。为得到这个结果，对 $\sum_{k=1}^{n-1}k\ln k$ 应用欧拉求和公式。如果我们以

$$\exp(-B_4/(2\cdot3\cdot4n^2) - \cdots - B_{2t}/((2t-2)(2t-1)(2t)n^{2t-2}) + O(1/n^{2t}))$$

来乘上边的答案，则得到更精确的公式。数 A 是“Glaisher 常数”1.2824271…[*Messenger of Math*. **7** (1877),43～47]，可以证明它等于 $e^{1/12-\zeta'(-1)} = (2\pi e^{\gamma-\zeta'(2)/\zeta(2)})^{1/12}$[de Bruijn, *Asymptotic Methods in Analysis*, §3.7]。

8. 例如，我们有 $\ln(an^2+bn) = 2\ln n + \ln a + \ln(1+b/(an))$。于是求得第一个问题的答案是 $2an^2\ln n + a(\ln a-1)n^2 + 2bn\ln n + bn\ln a + \ln n + b^2/(2a) + \frac{1}{2}\ln a + \sigma + (3a-b^2)b/(6a^2n) + O(n^{-2})$。当我们计算量 $\ln(cn^2)! - \ln(cn^2-n)! - n\ln c - \ln n^2! + \ln(n^2-n)! = (c-1)/(2c) - (c-1)(2c-1)/(6c^2n) + O(n^{-2})$ 时出现大量的抵消。因此答案是

$$e^{(c-1)/(2c)}\left(1 - \frac{(c-1)(2c-1)}{6c^2n}\right)(1+O(n^{-2}))$$

碰巧，$\binom{cn^2}{n}/c^n\binom{n^2}{n}$ 可以写作 $\prod_{j=1}^{n-1}(1+\alpha j/(n^2-j))$，其中 $\alpha = 1-1/c$。

9. (a) 我们有 $\ln(2n)! = \left(2n+\frac{1}{2}\right)\ln 2n - 2n + \sigma + \frac{1}{24n} + O(n^{-3})$，以及 $\ln(n!)^2 = (2n+1)\cdot\ln n - 2n + 2\sigma + \frac{1}{6n} + O(n^{-3})$；因此 $\binom{2n}{n} = \exp(2n\ln 2 - \frac{1}{2}\ln\pi n - \frac{1}{8n} + O(n^{-3})) = z^{2n}\cdot(\pi n)^{-1/2}\left(1 - \frac{1}{8}n^{-1} + \frac{1}{128}n^{-2} + O(n^{-3})\right)$。(b)由于 $\binom{2n}{n} = 2^{2n}\binom{n-1/2}{n}$ 和 $\binom{n-1/2}{n} = \Gamma(n+1/2)/(n\Gamma(n)\Gamma(1/2)) = n^{-1}n^{\overline{1/2}}/\sqrt{\pi}$，从 1.2.11.1-(16)我们得到相同的结果，因为

$$\left[{1/2 \atop 1/2}\right] = 1,\ \left[{1/2 \atop -1/2}\right] = \binom{1/2}{2} = -\frac{1}{8},\ \left[{1/2 \atop -3/2}\right] = \binom{1/2}{4} + 2\binom{3/2}{4} = \frac{1}{128}$$

方法(b)说明了为什么在

$$\binom{2n}{n} = \frac{2^{2n}}{\sqrt{\pi n}}\left(1 - \frac{n^{-1}}{8} + \frac{n^{-2}}{128} + \frac{5n^{-3}}{1024} - \frac{21n^{-4}}{32768} - \frac{399n^{-5}}{262144} + \frac{869n^{-6}}{4194304} + O(n^{-7})\right)$$

中的分母全是 2 的幂[Knuth 和 Vardi, *AMM* **97** (1990), 629 ~ 630]。

1.2.11.3 小节

1. 分部积分。

2. 在积分中代入 e^{-t}的级数。

3. 见等式 1.2.9-(11)和习题 1.2.6-48。

4. $1+1/u$ 作为 v 的函数是有界的,因为当 v 从 r 趋向无穷时,它趋于 0。以 M 代替它,得到的积分为 Me^{-rx}。

5. $f''(x)=f(x)((n+1/2)(n-1/2)/x^2-(2n+1)/x+1)$在点 $r=n+1/2-\sqrt{n+1/2}$处改变符号,所以 $|R|=O(\int_0^n|f''(x)|dx)=O(\int_0^r f''(x)dx-\int_r^n f''(x)dx)=O(f'(n)-2f'(r)+f'(0))=O(f(n)/\sqrt{n})$。

6. 它是 $n^{n+\beta}\exp((n+\beta)(\alpha/n-\alpha^2/2n^2+O(n^{-3})))$,等等。

7. 被积函数作为 x^{-1}的幂级数,有着如 $O(u^{2n})$那样的 x^{-n}的系数。在积分之后,x^{-3}中的项是 $Cu^7/x^3=O(x^{-5/4})$,等等。为了得到答案中的 $O(x^{-2})$,我们可以抛弃具有 $4m-n\geq 9$ 的诸项 u^n/x^m。因此,求积 $\exp(-u^2/2x)\exp(u^3/3x^2)\cdots$的展开最终导致答案

$$yx^{1/4}-\frac{y^3}{6}x^{-1/4}+\frac{y^5}{40}x^{-3/4}+\frac{y^4}{12}x^{-1}-\frac{y^7}{336}x^{-5/4}-\frac{y^6}{36}x^{-3/2}+\left(\frac{y^9}{3456}-\frac{y^5}{20}\right)x^{-7/4}+O(x^{-2})$$

8. (由 Miclós Simonovits 给出的解)如果 x 足够大,我们有 $|f(x)|<x$。令 $R(x)=\int_0^{f(x)}(e^{-g(u,x)}-e^{-h(u,x)})du$ 是两个给定积分之间的差,其中 $g(u,x)=u-x\ln(1+u/x)$ 和 $h(u,x)=u^2/2x-u^3/3x^2+\cdots+(-1)^m u^m/mx^{m-1}$。注意当 $|u|<x$ 时,$g(u,x)\geq 0$ 和 $h(u,x)\geq 0$;而且 $g(u,x)=h(u,x)+O(u^{m+1}/x^m)$。

按照均值定理,对于 a 和 b 之间的某个 c,有 $e^a-e^b=(a-b)e^c$。因此当 $a,b\leq 0$ 时 $|e^a-e^b|\leq|a-b|$。由此得出

$$|R(x)|\leq\int_{-|f(x)|}^{|f(x)|}|g(u,x)-h(u,x)|du=O\left(\int_{-Mx^r}^{Mx^r}\frac{u^{m+1}du}{x^m}\right)=O(x^{(m+2)r-m})=O(x^{-s})$$

9. 我们可以假定 $p\neq 1$,因为 $p=1$ 已由定理 A 给出。我们也可以假定 $p\neq 0$,因为该情况是不重要的。

情况 1:$p<1$。做代换 $t=px(1-u)$,因而 $v=-\ln(1-u)-pu$。我们有 $dv=((1-p+pu)/(1-u))du$,所以对于$0\leq u\leq 1$ 这个变换是单调的,而且我们得到形如

$$\int_0^\infty xe^{-xv}dv\left(\frac{1-u}{1-p+pu}\right)$$

的积分。由于带圆括弧的量是$(1-p)^{-1}(1-v(1-p)^{-2}+\cdots)$,答案因此是

$$\frac{p}{1-p}(pe^{1-p})^x\frac{e^{-x}x^x}{\Gamma(x+1)}\left(1-\frac{1}{(p-1)^2x}+O(x^{-2})\right)$$

情况 2:$p>1$。这是 $1-\int_{px}^\infty()$。在后边的积分中,做代换 $t=px(1+u)$,然后 $v=pu-\ln(1+u)$,而且如情况 1 那样进行。答案结果是等于情况 1 的同一公式加 1。注意 $pe^{1-p}<1$,所以 $(pe^{1-p})^x$ 非常之小。

习题 11 的答案给出求解这个问题的另一种方法。

10. $\frac{p}{p-1}(pe^{1-p})^x e^{-x}x^x\left(1-e^{-y}-\frac{e^{-y}(e^y-1-y-y^2/2)}{x(p-1)^2}+O(x^{-2})\right)$。

11. 首先，$xQ_x(n)+R_{1/x}(n)=n!(x/n)^n e^{n/x}$推广了(4)。其次，我们有 $R_x(n)=n!(e^x/nx)^n\gamma(n,nx)/(n-1)!$，推广了(9)。由于 $a\gamma(a,x)=\gamma(a+1,x)+e^{-x}x^a$，也可以写 $R_x(n)=1+(e^x/nx)^n\gamma(n+1,nx)$，并把这个问题同习题 9 关联起来。而且，通过使用等式 1.2.9-(27)和(28)我们可以直接处理 $Q_x(n)$和 $R_x(n)$，以便推导涉及斯特林数的级数展开：

$$2+xQ_x(n)=\sum_{k\geq 0}x^k n^{\underline{k}}/n^k=\sum_{k,m}\frac{(-1)^m}{n^m}\left[{k \atop k-m}\right]x^k$$

$$R_x(n)=\sum_{k\geq 0}x^k n^k/(n+1)^{\overline{k}}=\sum_{k,m}\frac{(-1)^m}{n^m}\left\{{k+m \atop k}\right\}x^k$$

当 $|x|<1$ 时，对于固定的 m，对 k 的求和收敛，而且 $|x|>1$ 是我们可以使用 $Q_x(x)$和 $R_{1/x}(x)$之间的关系；这就导致公式

$$Q_x(n)=\frac{1}{1-x}-\frac{x}{(1-x)^3 n}+\cdots+\frac{(-1)^m q_m(x)}{(1-x)^{2m+1}n^m}+O(n^{-1-m}),$$

$$R_x(n)=\frac{1}{1-x}-\frac{x}{(1-x)^3 n}+\cdots+\frac{(-1)^m r_m(x)}{(1-x)^{2m+1}n^m}+O(n^{-1-m}),\text{如果 } x<1;$$

$$Q_x(n)=\frac{n!\,x^{n-1}e^{n/x}}{n^n}+\frac{1}{1-x}-\frac{x}{(1-x)^3 n}+\cdots+\frac{(-1)^m q_m(x)}{(1-x)^{2m+1}n^m}+O(n^{-1-m}),$$

$$R_x(n)=\frac{n!\,e^{nx}}{n^n x^n}+\frac{1}{1-x}-\frac{x}{(1-x)^3 n}+\cdots+\frac{(-1)^m r_m(x)}{(1-x)^{2m+1}n^m}+O(n^{-1-m}),\text{ 如果 } x>1$$

这里

$$q_m(x)=\left\langle\!\!\left\langle {m \atop 0} \right\rangle\!\!\right\rangle x^{2m-1}+\left\langle\!\!\left\langle {m \atop 1} \right\rangle\!\!\right\rangle x^{2m-2}+\cdots$$

和

$$r_m(x)=\left\langle\!\!\left\langle {m \atop 0} \right\rangle\!\!\right\rangle x+\left\langle\!\!\left\langle {m \atop 1} \right\rangle\!\!\right\rangle x^2+\cdots$$

是系数为"第二阶欧拉数"的多项式[*CMath* §6.2；请见 L. Carlitz, *Proc. Amer. Math. Soc.* **16** (1965), 248～252。] $x=-1$ 的情况是有点棘手的，但可通过连续性来处理，因为由 $O(n^{-1-m})$所意味的界当 $x<0$ 时是与 x 无关的。注意到 $R_{-1}(n)-Q_{-1}(n)=(-1)^n n!/e^n n^n\approx(-1)^n\cdot\sqrt{2\pi n}/e^{2n}$非常小是很有趣的。

12. $\gamma\left(\frac{1}{2},\frac{1}{2}x^2\right)/\sqrt{2}$。

13. 请见 P. Flajolet, P. Grabner, P. Kirschenhofer 及 H. Prodinger, *J. Computational and Applied Math.* **58** (1995), 103～116。

15. 以二项式定理展开被积函数，我们求得积分为 $1+Q(n)$。

16. 把 $Q(k)$写作和式，并利用等式 1.2.6-(53)交换求和的次序。

17. $S(n)=\sqrt{\pi n/2}+\frac{2}{3}-\frac{1}{24}\sqrt{\pi/2n}-\frac{4}{135}n^{-1}+\frac{49}{1152}\sqrt{\pi/2n^3}+O(n^{-2})$。[注意 $S(n+1)+P(n)=\sum_{k\geq 0}k^{n-k}k!/n!$，而 $Q(n)+R(n)=\sum_{k\geq 0}n!/k!\,n^{n-k}$。]

18. 令 $S_n(x,y)=\sum_k\binom{n}{k}(x+k)^k(y+n-k)^{n-k}$。于是对于 $n>0$,由阿贝尔公式 1.2.6-(16),我们有 $S_n(x,y)=x\sum_k\binom{n}{k}(x+k)^{k-1}(y+n-k)^{n-k}+n\sum_k\binom{n-1}{k}(x+1+k)^k(y+n-1-k)^{n-1-k}=(x+y+n)^n+nS_{n-1}(x+1,y)$;结果 $S_n(x,y)=\sum_k\binom{n}{k}k!\,(x+y+n)^{n-k}$。[这个公式是由柯西给出的,他使用留数计算证明了该公式;请见他的*Œuvres* (2) **6**,62~73。]所述的和因此分别等于 $n^n(1+Q(n))$和$(n+1)^nQ(n+1)$。

19. 假设对于所有 $n\geq N$, C_n 存在,且对于 $0\leq x\leq r$, $|f(x)|\leq Mx^{\alpha}$。令 $F(x)=\int_r^x e^{-Nt}f(t)\,dt$。于是,当 $n>N$ 时,我们有

$$|C_n|\leq\int_0^r e^{-nx}|f(x)|\,dx+\left|\int_r^{\infty}e^{-(n-N)x}e^{-Nx}f(x)\,dx\right|\leq$$
$$M\int_0^r e^{-nx}x^{\alpha}dx+(n-N)\left|\int_r^{\infty}e^{-(n-N)x}F(x)\,dx\right|\leq$$
$$M\int_0^{\infty}e^{-nx}x^{\alpha}dx+(n-N)\sup_{x\geq r}|F(x)|\int_r^{\infty}e^{-(n-N)x}dx=$$
$$M\,\Gamma(\alpha+1)n^{-1-\alpha}+\sup_{x\geq r}|F(x)|e^{-(n-N)r}=O(n^{-1-\alpha})$$

[E. W. Barnes,*Phil. Trans.* **A206** (1906),249~297;G. N. Watson,*Proc. London Math. Soc.* **17** (1918),116~148。]

20.[C. C. Rousseau,*Applied Math. Letters* **2** (1989),159~161。]通过代换 $u=x-\ln(1+x)$并令 $g(u)=\frac{dx}{du}$,得到 $Q(n)+1=n\int_0^{\infty}e^{-nx}(1+x)^n dx=n\int_0^{\infty}e^{-n(x-\ln(1+x))}dx=n\int_0^{\infty}e^{-nu}g(u)\,du$。注意当 u 充分小时,$x=\sum_{k=1}^{\infty}c_k(2u)^{k/2}$。因此 $g(u)=\sum_{k=1}^{m-1}kc_k(2u)^{k/2-1}+O(u^{m/2-1})$,因此我们可以应用 Watson 引理到 $Q(n)+1-n\int_0^{\infty}e^{-nu}\sum_{k=1}^{m-1}kc_k(2u)^{k/2-1}du$。

1.3.1 小节

1. 4;每个字节将含有 $3^4=81$ 个不同的值。
2. 5,因为有 5 个字节总是足够了,但 4 个不够。
3. (0:2.); (3:3); (4:4); (5:5)。
4. 假定变址寄存器 4 含有大于或等于 2000 的值,使得经变址之后得到有效的内存地址。
5. “DIV - 80,3(0:5)”或者简单地“DIV - 80,3”。

6. (a) rA←

-	5	1	2000	15

。(b) rI2←200。(c) rX←

+	0	0	5	1	?

。(d) 无定义;因为我们不能把这样大的值装入变址寄存器中。(e) rX←

-	0	0	0	0	0

。

7. 运算之前令 $n=|\text{rAX}|$ 为寄存器 A 和 X 的数值,令 $d=|V|$ 为除数的数值。经运算之后,rA 的量为$\lfloor n/d\rfloor$,而且 rX 的量为 $n \bmod d$。以后 rX 的符号为 rA 先前的符号;如果 rA 和 V 先前的符号相同,则以后 rA 的符号为 +,否则为 -。

以另一种方法来表达:如果 rA 和 V 的符号相同,则 rA←$\lfloor$AX/V$\rfloor$和 rX←rAX mod V。否则 rA←$\lceil$rAX/V$\rceil$和 rX←rAX mod - V。

8. rA← | + | 0 | 617 | 0 | 1 | ；rX← | − | 0 | 0 | 0 | 1 | 1 |。

9. ADD，SUB，DIV，NUM，JOV，JNOV，INCA，DECA，INCX，DECX。

10. CMPA，CMP1，CMP2，CMP3，CMP4，CMP5，CMP6，CMPX。（还有对于浮点的 FCMP。）

11. MOVE，LD1，LD1N，INC1，DEC1，ENT1，ENN1。

12. INC3 0，3。

13. "JOV 1000"除了执行时间之外毫无差别。"JNOV 1001"在大多数情况下造成 rJ 的不同设置。"JNOV 1000"造成异乎寻常的差别，因为它可能使计算机死锁于无限的循环之中。

14. NOP 不带任何东西；ADD，SUB 有 F＝(0:0)或地址等于 *（指令的位置）以及 F＝(3:3)；HLT（依赖于如何翻译本习题的语句）；地址和变址为零的任何移位；SLC 或 SRC 有 0 变址和 10 的倍数的地址；有 F＝0 的 MOVE；JSJ * ＋1；有地址和变址为零的任何 INC 或 DEC 指令。但"ENT1 0，1"不总是空操作，因为它可以把 rI1 从 −0 变为 ＋0。

15. 70；80；120。（块大小的 5 倍。）

16. (a) STZ 0；ENT1 1；MOVE 0(49)；MOVE 0(50)。如果已经知道字节大小等于 100，则将仅仅需要一条 MOVE 指令，但我们并未获许对字节大小做出假定。(b)用 100 条 STZ 指令。

17. (a) STZ 0，2； DEC2 1； J2NN 3000。

(b)
```
        STZ   0
        ENT1  1
        JMP   3004
(3003)  MOVE  0(63)
(3004)  DEC2  63
        J2P   3003
        INC2  63
        ST2   3008(4:4)
(3008)  MOVE  0
```
（稍快但很怪的程序使用 993 个 STZ：JMP 3995；STZ 1，2；STZ 2，2；…；STZ 993，2；J2N 3999；DEC2 993；J2NN 3001；ENN1 0，2；JMP 3000，1。）

18. （如果你已经正确地遵循这些指令的话，则在执行 ADD 时将出现溢出，而且之后在寄存器 A 中有负 0。）答案：溢出开关被置位，比较指示器置成 EQUAL，rA 置成 | − | 30 | 30 | 30 | 30 | 30 |，rX 置成 | − | 31 | 30 | 30 | 30 | 30 |，rI1 置成 ＋3，而且内存单元 0001，0002 置成 ＋0。（除非程序本身在单元 0000 处开始。）

19. $42u=(2+1+2+2+1+1+1+2+2+1+2+2+3+10+10u)$。

20. （由 H. Fukuoka 给出的解。）
```
(3991)  ENT1  0
        MOVE  3995    (MOVE 的标准的 F 为 1)
(3993)  MOVE  0(43)   (3999 = 93 乘以 43)
        JMP   3993
(3995)  HLT   0
```

21. (a) 不能，除非它可以通过外部手段置成 0（见习题 26 的"GO 按钮"），因为程序仅能通过由地址 $N-1$ 的转移来置 rJ←N。

```
(b)     LDA   -1,4
        LDX   3004
        STX   -1,4
        JMP   -1,4
(3004)  JMP   3005
(3005)  STA   -1,4
```

22. 极小时间：如果 b 是字节大小，则 $|X^{13}| < b^5$ 之假定意味着 $X^2 < b$，所以 X^2 可包含在一个字节内。下面是 Y. N. Patt 利用这一事实给出的巧妙解答。rA 符号是 X 的符号。

```
(3000)  LDA   2000
        MUL   2000(1:5)
        STX   3500(1:1)
        SRC   1
        MUL   3500
        STA   3501
        ADD   2000
        MUL   3501(1:5)
        STX   3501
        MUL   3501(1:5)
        SLAX  1
        HLT   0
(3500)  NOP   0
(3501)  NOP   0
```

<table>
<tr><th colspan="5">rA</th><th colspan="5">rX</th></tr>
<tr><td>X^2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="2">X^4</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="2">X^4</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="2">X^4</td><td>0</td><td>0</td><td>X</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="4">X^8</td><td>0</td><td colspan="2">X^5</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="4">X^8</td><td>0</td><td colspan="2">X^5</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td colspan="5">X^{13}</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="5">X^{13}</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

空间 = 14；时间 = $54u$，没有计算 HLT 的时间。

按照在 4.6.3 小节提出的理论，至少有 5 个乘法是“必要的”，而这个程序却仅用了 4 个！而且事实上在下面还有一个甚至更好的解。

极小空间：

```
(3000)  ENT4  12
        LDA   2000
(3002)  MUL   2000
        SLAX  5
        DEC4  1
        J4P   3002
        HLT   0
```

空间 = 7，时间 = $171u$。

真正的极小时间：如 R. W. Floyd 所指出的，这些条件意味着 $|X| \leqslant 5$，所以通过访问一个表，可以达到极小的执行时间：

```
(3000)  LD1   2000
        LDA   3500,1
        HLT   0
(3495)  (-5)^13
(3496)  (-4)^13
```

⋮

(3505) (+5)[13]

空间 = 14;时间 = $4u$。

23. 由 R. D. Dixon 给出的下列解答,看来满足所有条件:

```
(3000) ENT1  4      DEC1   1
(3001) LDA   200    J1NN   3001
       SRA   0,1    SLAX   5
       SRAX  1      HLT    0   ▌
```

24. (a) DIV 3500,其中 3500 = | + | 1 | 0 | 0 | 0 | 0 |。

(b)SRC 4; SRA 1; SLC 5。

25. **一些思路**:(a) 明显的事情,如更快速的存储器,更多的输入输出设备。(b) I 字段可用作 J 寄存器变址,和/或多重变址(确定两个不同的变址寄存器),和/或“间接编址”(习题 2.2.2-3,4,5)。(c) 变址寄存器和 J 寄存器可以扩充为完全的 5 个字节;因此具有更高地址的内存位置仅可通过变址来访问,但是如果像在(b)中那样有多重变址可利用,则这并非不可容忍。(d) 像在习题 1.4.4-18 中那样,利用负的内存地址,可以加上中断的能力。(e)在负的内存地址下,可以加上“实时钟”。(f)按二进位的操作,根据寄存器的偶或奇进行转移,以及二进制的移位可以加到 MIX 的二进制版本中(例如,请见习题 2.5-28,5.2.2-12 以及 6.3-9;以及程序 4.5.2B,6.4-(24),以及 7.1 节)。(g) 一个“执行”命令,指的是执行在单元 M 中的指令,可以有 C = 5 的另一个变形。(h) C = 48,⋯,55 的另一个变形可以置 C1←寄存器:M。

26. 使用(2:5)字段来获得卡片的 7 ~ 10 列是有吸引力的,但这不可能做到,因为 2 × 8 + 5 = 21。为使程序员易于理解,这里以符号语言来给出它,预先利用 1.3.2 小节的内容。

```
      BUFF   EQU    29              缓冲区区域是 0029 ~ 0044   穿在卡片上的字符
             ORIG   0
 00   LOC    IN     16(16)          读入第二张卡片             ␣O␣O6
 01   READ   IN     BUFF(16)        读下一张卡片               ␣Z␣O6
 02          LD1    0(0:0)          rI1←0                      ␣␣␣␣I
 03          JBUS   *(16)           等待输入完成               ␣C␣04
 04          LDA    BUFF+1          rA←列 6 ~ 10               ␣0␣EH
 05   =1=    SLA    1                                          ␣A␣␣F
 06          SRAX   6               rAX←列 7 ~ 10              ␣F␣CF
 07   =30=   NUM    30                                         ␣O␣␣E
 08          STA    LOC             LOC←启始单元               ␣␣␣EU
 09          LDA    BUFF+1(1:1)                                ␣0␣IH
 10          SUB    =30=(0:2)                                  ␣G␣BB
 11   LOOP   LD3    LOC             rI3←LOC                    ␣␣␣EJ
 12          JAZ    0,3             如果传输卡片,则转移        ␣␣CA.
 13          STA    BUFF            BUFF←计数                  ␣Z␣EU
 14          LDA    LOC                                        ␣␣␣EH
 15          ADD    =1=(0:2)                                   ␣E␣BA
 16          STA    LOC             LOC←LOC+1                  ␣␣␣EU
```

```
         LDA   BUFF+3,1(5:5)                      ⊔2A - H
         SUB   =25=(0:2)                          ⊔S⊔BB
         STA   0.3(0:0)       存储符号            ⊔⊔C⊔U
         LDA   BUFF+2,1                           ⊔ 1 AEH
         LDX   BUFF+3,1                           ⊔ 2 AEN
=25=     NUM   25                                 ⊔V⊔⊔E
         STA   0,3(1:5)       存储值              ⊔ ⊔ CLU
         MOVE  0,1(2)         rI1←rI1+2,(!)       ⊔ ⊔ ABG
         LDA   BUFF                               ⊔Z⊔ EH
         SUB   =1=(0:2)       计数器减 1          ⊔B⊔ BB
         JAP   LOOP           重复直到计数为零为止  ⊔J⊔B .
         JMP   READ           现在读一张新的卡片   ⊔A⊔⊔9
```

1.3.2 小节

1. ENTX 1000; STX X。

2. 行 03 中指令 STJ 设置这一地址。(通过“ * ”来表示这种指令的地址是方便的,这既因为写起来简单,也因为它提供了程序中的一个出错条件之可识别的测试,即由于某些疏忽而未曾正确地输入子程序的情况。某些人喜欢用“ *-* ”。)

3. 从磁带设备 0 读入 100 个字;把这些字中的极大值与最后一个字交换;把剩下的 99 个中的极大值与这些中之最后一个交换;等等。最后这 100 个字将完全地排序为递增的序列。然后把结果写到磁带设备 1 上。(试与 5.2.3S 中的算法作比较。)

4. 非 0 单元:

3000:	+	0000	00	18	35	3013:	+	0001	00	00	51
3001:	+	2051	00	05	09	3014:	+	3008	00	06	39
3002:	+	2050	05	05	10	3015:	+	3003	00	00	39
3003:	+	0001	00	00	49	3016:	+	1995	00	18	37
3004:	+	0499	01	05	26	3017:	+	2035	00	02	52
3005:	+	3016	00	01	41	3018:	−	0050	00	02	53
3006:	+	0002	00	00	50	3019:	+	0501	00	00	53
3007:	+	0002	00	02	51	3020:	−	0001	05	05	08
3008:	+	0000	00	02	48	3021:	+	0000	00	01	05
3009:	+	0000	02	02	55	3022:	+	0000	04	12	31
3010:	−	0001	03	05	04	3023:	+	0001	00	01	52
3011:	+	3006	00	01	47	3024:	+	0050	00	01	53
3012:	−	0001	03	05	56	3025:	+	3020	00	02	45

3026:	+	0000		04	18	37
3027:	+	0024		04	05	12
3028:	+	3019		00	00	45
3029:	+	0000		00	02	05
0000:	+					2
1995:	+	06	09	19	22	23
1996:	+	00	06	09	25	05
1997:	+	00	08	24	15	04

1998:	+	19	05	04	00	17
1999:	+	19	09	14	05	22
2024:	+					2035
2049:	+					2010
2050:	+					3
2051:	−					499

(最后两个可以交换,相应地改变 3001 和 3002)

5.每个 OUT 都等待以前的打印机操作完成(从另一个缓冲区)。

6.(a) 如果 n 不是素数,则由定义 n 有一个因子 d 满足 $1<d<n$。如果 $d>\sqrt{n}$,则 n/d 是满足 $1<n/d<\sqrt{n}$的因子。(b)如果 N 不是素数,则 N 有一个素因子 d 满足 $1<d\leq\sqrt{\mathrm{N}}$。这个算法已经验证了 N 没有小于等于 $p=$ PRIME[K]的素因子;而且 $\mathrm{N}=p\mathrm{Q}+\mathrm{R}<p\mathrm{Q}+p\leq p^2+p<(p+1)^2$。因此 N 的任何素因子 $>p+1>\sqrt{\mathrm{N}}$。

我们还必须证明,当 N 为素数时,将有一个充分大的素数小于 N,即是,第 $k+1$ 个素数 p_{k+1} 小于 $p_k^2+p_k$;否则当我们需要它很大时,K 将超过 J 而且 PRIME[K]将为零。这由"Bertrand 假设"推出:如果 p 是素数,则有一个小于 $2p$ 的更大的素数。

7.(a)它访问行 29 的单元。(b) 这个程序于是将失败;行 14 将访问行 15 而不是行 25;行 24 将访问行 15 而不是行 12。

8. 打印 100 行。如果把这些行上的 12000 个字符衔接起来,那它们将达到相当之远,而且将组成由 5 个空白,其后紧跟 5 个 A,后边接着 10 个空白,其后紧跟 5 个 A,后边接着 15 个空白,…,后边接着 $5k$ 个空白,其后紧跟 5 个 A,后边接着 $5(k+1)$个空白,…,直到打印完 12 000 个字符为止。倒数第三行以 AAAAA 和 35 个空白结尾;最后两行全是空白。总的效果是操作码(OP)的技巧之一。

9. 在下列表格中,每个条目的(4:4)字段存有极大的 F 的设置;(1:2)字段是适当的正确性检验程序的位置。

```
B       EQU   1(4:4)
BMAX    EQU   B-1
UMAX    EQU   20
TABLE   NOP   GOOD(BMAX)
        ADD   FLOAT(5:5)
        SUB   FLOAT(5:5)
        MUL   FLOAT(5:5)
        DIV   FLOAT(5:5)
        HLT   GOOD
        SRC   GOOD
        MOVE  MEMORY(BMAX)
        LDA   FIELD(5:5)
        ...
        STZ   FIELD(5:5)
        JBUS  MEMORY(UMAX)
        IOC   GOOD(UMAX)
        IN    MEMORY(UMAX)
        OUT   MEMORY(UMAX)
        JRED  MEMORY(UMAX)
        JLE   MEMORY
        JANP  MEMORY
        ...
```

```
      JXNP  MEMORY                              FLOAT  CMPA  VALID (4:4)   F=6在算术op
      ENNA  GOOD                                       JE    MEMORY          中是允许的
      …                                         FIELD  ENTA  0
      ENNX  GOOD                                       LDX   INST (4:4)    这是校验有效的
      CMPA  FLOAT (5:5)                                DIV   =9=             部分字段的一
      CMP1  FIELD (5:5)                                STX   * +1(0:2)       项技巧性方
      …                                                INCA  0               法
      CMPX  FILED (5:5)                                CMPA  =5=
BEGIN LDA   INST                                       JG    BAD
      CMPA  VALID (3:3)                         MEMORY LDX   INST (3:3)
      JG    BAD                  I字段>6?              JXNZ  GOOD          如果I=0,则保证
      LD1   INST (5:5)                                 LDX   ISNT (0:2)      这个地址是
      DEC1  64                                         JXN   BAD             一个有效的
      J1NN  BAD                  C字段≥64?             CMPX  =3999=          存储单元
      CMPA  TABLE+64,1(4:4)                            JLE   GOOD
      JG    BAD                  F字段>Fmax?           JMP   BAD
      LD1   TABLE+64,1(1:2)      转到特殊的程   VALID  CMPX  3999,6(6)     ▌
      JMP   0,1                    序
```

10. 对这个问题要领会的是,在一行或一列中可能有若干个位置上出现极小(极大)值,而且每一个都是潜在的鞍点。

解法 1　在这个解法中,我们依次地跑遍每一行,并造一份其中出现有行极小的所有列表,然后校验这份表的每一列,看看行的极小是否也是列的极大。rX≡当前的极小;rI1 跟踪整个矩阵,从 72 列跑到 0 列,除非找到了一个鞍点;rI2≡rI1 的列下标;rI3≡极小的表的大小。注意,在所有情况下,一个循环的终止条件是变址寄存器≤0。

```
 * SOLUTION  1
A10       EQU     1008        a10的位置
LIST      EQU     1000
START     ENT1    9*8         在右下角处开始
ROWMIN    ENT2    8           现在rI1是在行的第8列
2H        LDX     A10,1       行极小的候选者
          ENT3    0           表空
4H        INC3    1
          ST2     LIST,3      在表中置列下标
1H        DEC1    1           左移1
          DEC2    1
          J2Z     COLMAX      对于行完了吗?
3H        CMPX    A10,1
          JL      1B          rX仍是极小?
          JG      2B          新的极小?
          JMP     4B          记住另一个极小出现
```

```
COLMAX  LD2    LIST,3      从表中得到列
        INC2   9*8-8
1H      CMPX   A10,2
        JL     NO          行最小<列元素?
        DEC2   8
        J2P    1B          对于列做完了吗?
YES     INC1   A10+8,2     是;rI1←鞍点的地址
        HLT
NO      DEC3   1           表空了吗?
        J3P    COLMAX      否;再试
        J1P    ROWMIN      所有的行都已试完?
        HLT                是;rI1=0,无鞍点 ▌
```

解法 2 引进数学,给出不同的算法。

定理 *命 $R(i)=\min_j a_{ij}$, $C(j)=\max_i a_{ij}$。元素 $a_{i_0j_0}$是一个鞍点,当且仅当 $R(i_0)=\max_i R(i)=C(j_0)=\min_j C(j)$。*

证明 如果 $a_{i_0j_0}$是一个鞍点,则对于任何固定的 i, $R(i_0)=C(j_0)\geqslant a_{ij_0}\geqslant R(i)$;所以 $R(i_0)=\max_i R(i)$。类似地 $C(j_0)=\min_j C(j)$。反之,对于所有的 i 和 j 有 $R(i)\leqslant a_{ij}\leqslant C(j)$;因此 $R(i_0)=C(j_0)$意味着 $a_{i_0j_0}$是鞍点。 ▌

(这个证明表明,我们总有 $\max_i R(i)\leqslant\min_j C(j)$。所以不存在鞍点,当且仅当所有的 R 都小于所有的 C。)

按照这个定理,只需求出最小的列的极大值就行了,然后来查找相等的行极小值。在阶段 1,rI1≡列下标;rI2 跑遍矩阵。在阶段 2,rI1≡可能的答案;rI2 跑遍矩阵;rI3≡行下标乘以 8;rI4≡列下标。

```
* SOLUTION  2
CMAX    EQU    1000
A10     EQU    CMAX+8
PHASE1  ENT1   8           在列 8 开始
3H      ENT2   9*8-8,1
        JMP    2F
1H      CMPX   A10,2       rX 仍是极小?
        JGE    *+2
2H      LDX    A10,2       列中的新极小
        DEC2   8
        J2P    1B
        STX    CMAX+8,2    存列极大
        J2Z    1F          第一次?
        CMPA   CMAX+8,2    rA 仍为 min max?
        JLE    *+2
```

```
1H      LDA    CMAX+8,2,2
        DEC1   1            左移一列
        J1P    3B
PHASE2  ENT3   9*8-8        这时 rA = min_j C(j)
3H      ENT2   8,3          准备寻找一行
        ENT4   8
1H      CMPA   A10,2        min_j C(j) > a[i,j]吗?
        JG     NO           这行中无鞍点
        JL     2F
        CMPA   CMAX,4       a[i,j] = C(j)吗?
        JNE    2F
        ENT1   A10,2        记住一个可能的鞍点
2H      DEC4   1            在行中向左移
        DEC2   1
        J4P    1B
        HLT                 找到了鞍点
NO      DEC3   8
        JMP    3B           试另一行
        ENT1   0
        HLT                 rI1 = 0;无鞍点  ▌
```

我们把此问题留给读者来想出更好的解法:此解法在阶段 1 中记录所有可能的这样的行,就是它们都是在阶段 2 中寻找的行的候选者。没有必要来寻找所有的行,仅仅是这样的 i_0 才是必要的:对于它 $C(j_0)=\min_j C(j)$意味着 $a_{i_0j_0}=C(j_0)$。通常至多只有一个这样的行。

在对于从{0,1,2,3,4}中随机地选择之元素的某些试验运行中,解法 1 近乎需要 $730u$ 时间来运行,而解法 2 大约要花 $530u$。给定一个全 0 的矩阵,解法 1 用 $137u$ 时间寻找出鞍点,而解法 2 用 $524u$ 的时间。

如果一个 $m\times n$ 矩阵有不同的元素,且 $m\geqslant n$,我们可以通过仅考察 $O(m+n)$个元素和进行$O(m\log n)$个辅助的操作就解决这个问题。请见 Bienstock, Chung, Fredman, Schäffer, Shor 及 Suri,*AMM* **98** (1991),418~419。

11. 假定一个 $m\times n$ 矩阵。(a) 由习题 10 答案中的定理,一个矩阵的所有鞍点有相同的值,所以(在我们关于诸元素不同的假定下)至多有一个鞍点。由对称性,所求的概率是 mn 乘以 a_{11} 为一鞍点的概率。后者为 $1/(mn)!$ 乘以对于 $a_{12}>a_{11},\cdots,a_{1n}>a_{11},a_{11}>a_{21},\cdots,a_{11}>a_{m1}$的排列数;这是 $1/(m+n-1)!$ 乘以 $m+n-1$ 个元素的排列的个数,其中第一个大于下面的$(m-1)$元素,并小于剩下的$(n-1)$个,即是$(m-1)!(n-1)!$。因此答案是

$$mn(m-1)!(n-1)!/(m+n-1)!=(m+n)\Big/\binom{m+n}{n}$$

在我们的情况下,这是 $17\Big/\binom{17}{8}$,即在 1430 中的仅有一次机会。(b) 在第二个假定之下,由于可以有多个鞍点,因此必须使用完全不同的方法;事实上,或者是整行或者是整列,必须完全由鞍点组成。这个概率等于:有一个鞍点具有值 0 的概率,加上有一个鞍点具有值 1 的概率。前者是至

少有一列 0 的概率;后者是至少有一行 1 的概率。答案是$(1-(1-2^{-m})^n)+(1-(1-2^{-n})^m)$;在我们的情况下,是 9247447796234036231/18446744073709551616,大约为 1/19.9。近似的答案为 $n2^{-m}+m2^{-n}$。

12. M. Hofri 和 P. Jacquet [*Algorithmica* **22** (1998),516~528]分析了 $m\times n$ 矩阵各项不同且次序随机时的情况。当 $m\to\infty$ 且 $n\to\infty$ 时,假定$(\log n)/m\to 0$,此时两个 MIX 程序的运行时间分别是$(6mn+5mH_n+8m+6+5(m+1)/(n-1))u+O\left((m+n)^2\Big/\binom{m+n}{m}\right)$和$(5mn+2nH_m+7m+7n+9H_n)u+O(1/n)+O((\log n)^2/m)$。

13.
```
* CRYPTANALYST PROBLEM (CLASSIFIED)
TAPE    EQU    20             输入设备号
TYPE    EQU    19             输出设备号
SIZE    EQU    14             输入块大小
OSIZE   EQU    14             输出块大小
TABLE   EQU    1000           计数表
        ORIG   TABLE          (除了对于
        CON    -1             空格和星
        ORIG   TABLE+46       号的项外,
        CON    -1             开始为0)
        ORIG   2000
BUF1    ORIG   *+SIZE         头一个缓冲区区域
        CON    -1             缓冲区末端的“标志”
        CON    *+1            对第二个缓冲区的访问
BUF2    ORIG   *+SIZE         第二个缓冲区
        CON    -1             “标志”
        CON    BUF1           对第一个缓冲区的访问
BEGIN   IN     BUF1(TAPE)     输入第一块
        ENT6   BUF2
1H      IN     0,6(TAPE)      输入下一块
        LD6    SIZE+1,6       在进行这个输入期间,准备
        ENT5   0,6              处理以前的输入
        JMP    4F
2H      INCA   1                                 ⎫  主循环
        STA    TABLE,1        更新表项           ⎪
3H      SLAX   1                                 ⎬ 应当运行得
        STA    *+1(2:2)       rI1←下一个字符     ⎪
        ENT1   0                                 ⎭ 尽可能地快
        LDA    TABLE,1
        JANN   2B             是通常字符吗?
        J1NZ   3F             是星号吗?
        JXP    3B             跳过一个空格
        INC5   1
```

```
4H       LDX      0,5              rX←五个字符
         JXNN     3B               如果不是一个标志则转移
         JMP      1B               对块做完了
3H       ENT1     1                开始游戏的最后阶段:rI1←“A”
2H       LDA      TABLE,1
         JANP     1F               跳过0答案
         CHAR                      转换成十进制
         JBUS     *(TYPE)          等候打字机就绪
         ST1      CHAR(1:1)
         STA      CHAR(4:5)
         STX      FREQ
         OUT      ANS(TYPE)        打出一个答案
1H       CMP1     =63=
         INC1     1                直到累计了63个
         JL       2B                 字符代码
         HLT
ANS      ALF                       输出缓冲区
         ALF
CHAR     ALF      CNN
FREQ     ALF      NNNNN
         ORIG     ANS+OSIZE        缓冲区的剩下部分为空白
         END      BEGIN            文字常数=63=到这里。 ▌
```

对于这个问题,输出缓冲不是所希望的,因为对于每一行的输出,它至多可以节省 $7u$ 的时间。关于字母频率的信息,见 Charles P. Bourne 和 Donald F. Ford,“A study of the statistics of letters in English words”,*Information and Control* **4** (1961),48~67。

14. 为使得这个问题更有挑战性,下列部分地由 J. Petolino 给出的解法使用尽可能多的技巧,为的是减少执行时间。读者还能否再挤出点时间来呢?[Udo Wermuth 确实挤出了额外的 $9u$,$10u$ 或 $12u$ 的时间,它们与年份有关。但这里略去了细节,因为 MIX 不久就会被取代了。]

```
* DATE OF EASTER
EASTER   STJ      EASTX
         STX      Y
         ENTA     0                E1.
         DIV      =19=
         STX      GMINUS1(0:2)
         LDA      Y                E2.
         MUL      =1//100+1=       (见下面)
         INCA     61
         STA      CPLOS60(1:2)
         MUL      =3//4+1=
         STA      XPLUS57(1:2)
```

```
CPLUS60   ENTA   *
          MUL    =8//25+1=          rA←Z+24
GMINUS1   ENT2   *                  E5.
          ENT1   1,2                rI1←G
          INC2   1,1
          INC2   0,2
          INC2   0,1
          INC2   0,2
          INC2   773,1              rI2←11G+773
XPLUS57   INCA   -*,2               rA←11G+Z-X+20+24·30(≥0)
          SRAX   5
          DIV    =30=               rX←E
          DECX   24
          JXN    4F
          DECX   1
          JXP    2F
          JXN    3F
          DEC1   11
          J1NP   2F
3H        INCX   1
2H        DECX   29                 E6.
4H        STX    20MINUSN(0:2)
          LDA    Y                  E4.
          MUL    =1//4+1=
          ADD    Y
          SUB    XPLUS57(1:2)       rA←D-47
20MINUSN  ENN1   *
          INCA   67,1               E7.
          SRAX   5                  rX←D+N
          DIV    =7=
          SLAX   5
          DECA   -4,1               rA←31-N
          JAN    1F                 E8.
          DECA   31
          CHAR
          LDA    MARCH
          JMP    2F
1H        CHAR
          LDA    APRIL
2H        JBUS   *(18)
          STA    MONTH
```

```
         STX    DAY(1:2)
         LDA    Y
         CHAR
         STX    YEAR
         OUT    ANS(18)         打印
EASTX    JMP    *
MARCH    ALF    MARCH
APRIL    ALF    APRIL
ANS      ALF
DAY      ALF    DD
MONTH    ALF    MMMMM
         ALF    ,
YEAR     ALF    YYYYY
         ORIG   *+20
BEGIN    ENTX   1950            "驱动"程序
         ENT6   1950-2000        使用
         JMP    EASTER           上述的
         INC6   1                子程序
         ENTX   2000,6
         J6NP   EASTER+1
         HLT
         END    BEGIN           ▌
```

在许多位置处由除法变成为乘法的严格证明可以以 rA 中的数不太大这一事实为基础。对于所有的字节大小这个程序都有效。

[要计算 1582 年以前的复活节,请见*CACM* **5** (1962),209~210。用于计算复活节日期的第一个系统算法是由 Aquitania 的 Victorius(公元 457 年)提出的复活节正经(canon paschalis)。有许多迹象表明,在中世纪时期的欧洲算术的惟一非平凡的应用是复活节日期的计算,因此这样的算法在历史上是有意义的。关于进一步的评述,请见 T. H. O'Beirne 所著的*Puzzles and Paradoxes* (London:Oxford University Press,1965),第 10 章;关于所有种类的面向日期的算法也请见由 E. M. Reingold 和 N. Dershowitz 写的书*Calendrical Calculations* (Cambridge Univ. Press,2001)。]

15. 第一个这样的年份是公元 10317 年,尽管对于 $0 \leqslant k \leqslant 10$ 在公元 $10108 + 19k$ 年误差几乎导致错误。

顺便说明,T. H. O'Beirne 指出复活节的日期以精确的 5 700 000 年的周期重复。Robert Hill 的计算表明最常出现的日期是 4 月 19 日(在每个周期里是 220 400 次),而最早且最不常见的是 3 月 22 日(27 550 次);最晚且次最不常规的是 4 月 25 日(42 000 次)。Hill 发现了关于这一奇怪事实的很好说明,即在这个周期里任何特定日子出现的次数总是 25 的倍数。

16. 以比例数 $R_n = 10^n r_n$ 来进行工作。于是 $R_n(1/m) = R$ 当且仅当 $10^n/\left(R+\frac{1}{2}\right) < m \leqslant 10^n/\left(R-\frac{1}{2}\right)$;于是我们求得 $m_h = \lfloor 2\cdot 10^n/(2R-1)\rfloor$。

```
* SUM OF  HARMONIC SERIES
```

```
BUF     ORIG    * + 24
START   ENT2    0
        ENT1    3           5 − n
        ENTA    20
OUTER   MUL     = 10 =
        STX     CONST       2·10^n
        DIV     = 2 =
        ENTX    2
        JMP     1F
INNER   STA     R
        ADD     R
        DECA    1
        STA     TEMP        2R − 1
        LDX     CONST
        ENTA    0
        DIV     TEMP
        INCA    1
        STA     TEMP        m_h + 1
        SUB     M
        MUL     R
        SLAX    5
        ADD     S
        LDX     TEMP
1H      STA     S           部分和
        STX     M           m = m_e
        LDA     M
        ADD     M
        STA     TEMP
        LDA     CONST
        ADD     M           计算 R = R_n(1/m) =
        SRAX    5           ⌊(2·10^n + m)/(2m)⌋
        DIV     TEMP
        JAP     INNER       R > 0?
        LDA     S           10^n S_n
        CHAR
        SLAX    0,1         简洁的格式
        SLA     1
        INCA    40          小数点
        STA     BUF,2
        STX     BUF + 1,2
        INC2    3
```

```
          DEC1    1
          LDA     CONST
          J1NN    OUTER
          OUT     BUF(18)
          HLT
          END     START ▌
```

在 65595u 加上输出时间中，输出是

```
0006.16      0008.449     0010.7509     0013.05363
```

（当 $m < 10^{n/2}\sqrt{2}$时直接计算 $R_n(1/m)$，而后应用所提议的例程将是更快的。）

17. 令 $N = \lfloor 2\cdot 10^n/(2m+1)\rfloor$。于是 $S_n = H_N + O(N/10^n) + \sum_{k=1}^{m}(\lfloor 2\cdot 10^n/(2k-1)\rfloor - \lfloor 2\cdot 10^n/(2k+1)\rfloor)k/10^n = H_N + O(m^{-1}) + O(m/10^n) - 1 + 2H_{2m} - H_m = n\ \ln 10 + 2\gamma - 1 + 2\ \ln\ 2 + O(10^{-n/2})$，如果我们用分部求和并置 $m \approx 10^{n/2}$的话。

顺便指出，以下的若干个值是 $S_6 = 15.356262$, $S_7 = 17.6588276$, $S_8 = 19.96140690$, $S_9 = 22.263991779$，以及 $S_{10} = 24.5665766353$；对于 S_{10}我们的近似值是≈ 24.566576621，它比预期的更为接近。

```
18.FAREY  STJ     9F          假设 rI1 包含 n，其中 n > 1
          STZ     X           x0←0
          ENTX    1
          STX     Y           y0←1
          STX     X+1         x1←1
          ST1     Y+1         y1←n
          ENT2    0           k←0
1H        LDX     Y,2
          INCX    0,1
          ENTA    0
          DIV     Y+1,2
          STA     TEMP        ⌊(y_k + n)/y_{k+1}⌋
          MUL     Y+1,2
          SLAX    5
          SUB     Y,2
          STA     Y+2,2       y_{k+2}
          LDA     TEMP
          MUL     X+1,2
          SLAX    5
          SUB     X,2
          STA     X+2,2       x_{k+2}
          CMPA    Y+2,2       检测是否 x_{k+2} < y_{k+2}
          INC2    1           k←k+1
          JL      1B          如果是，则继续
9H        JMP     *           从子程序离开 ▌
```

19. (a) 归纳法。(b) 令 $k \geqslant 0$ 及 $X = ax_{k+1} - x_k$, $Y = ay_{k+1} - y_k$,其中 $a = \lfloor (y_k + n)/y_{k+1} \rfloor$。由(a)部分和 $0 < Y \leqslant n$ 的事实,我们有 $X \perp Y$ 和 $X/Y > x_{k+1}/y_{k+1}$。因此如果 $X/Y \neq x_{k+2}/y_{k+2}$,则由定义,我们有 $X/Y > x_{k+2}/y_{k+2}$,但这意味着

$$\frac{1}{Yy_{k+1}} = \frac{Xy_{k+1} - Yx_{k+1}}{Yy_{k+1}} = \frac{X}{Y} - \frac{x_{k+1}}{y_{k+1}} =$$

$$\left(\frac{X}{Y} - \frac{x_{k+2}}{y_{k+2}}\right) + \left(\frac{x_{k+2}}{y_{k+2}} - \frac{x_{k+1}}{y_{k+1}}\right) \geqslant$$

$$\frac{1}{Yy_{k+2}} + \frac{1}{y_{k+1}y_{k+2}} = \frac{y_{k+1} + Y}{Yy_{k+1}y_{k+2}} > \frac{n}{Yy_{k+1}y_{k+2}} \geqslant \frac{1}{Yy_{k+1}}$$

历史点评:C. Haros 在 *J. de l'École Polytechnique* **4**, 11(1802), 364 ~ 368 中给出用于构造这样的序列的一个(更复杂的)规则;他的方法是正确的,但他的证明不适当。许多年后,地质学家 John Farey 独立地猜想 x_k/y_k 总是等于 $(x_{k-1} + x_{k+1})/(y_{k-1} + y_{k+1})$[*Philos. Magazine and Journal* **47** (1816), 385 ~ 386];过后不久柯西提供了证明[*Bull. Société Philomathique de Paris* (3) **3** (1816), 133 ~ 135],他给这个序列冠上 Farey 的名字。有关更多的它的有趣性质,请见 G. H. Hardy 和 E. M. Wright 的 *An Introduction to the Theory of Numbers*, 第 3 章。

20.
```
* TRAFFIC SIGNAL PROBLEM
BSIZE    EQU    1(4:4)                 字节大小
2BSIZE   EQU    2(4:4)                 两倍字节大小
DELAY    STJ    1F                     如果 rA 含有 n,则
         DECA   6                        这个子程序恰好
         DECA   2                        等候 max(n,7)u,
         JAP    *-1                      不包括转到
         JAN    *+2                      这个子程序
         NOP                             的转移时间
1H       JMP    *
FLASH    STJ    2F           4         这个子程序使适当的
         ENT2   8            5           DON'T WALK 灯闪亮
1H       LDA    =49991=      7
         JMP    DELAY        8
         DECX   0,1          9         使灯关闭
         LDA    =49996=      2
         JMP    DELAY        3
         INCX   0,1          4         "DON'T WALK"
         DEC2   1            1
         J2Z    1F           2         重复八次
         LDA    =49993=      4
         JMP    1B           5         回到同步
1H       LDA    =399992=     4         在出口之后置黄色 2u
         JMP    DELAY        5
2H       JMP    *            6
```

```
WAIT     JNOV   *            5             德尔玛变绿灯直到顺利通过
TRIP     INCX   BSIZE        6             德尔玛上的 DON'T WALK(不许通过)
         ENT1   2BSIZE       1
         JMP    FLASH        2             德尔玛灯闪
         LDX    BAMBER       8             在大街上黄灯
         LDA    =799995=     2
         JMP    DELAY        3             等候 8 秒
         LDX    AGREEN       5             大道上绿灯
         LAD    =799996=     2
         JMP    DELAY        3             在伯克利上灯闪
         INCX   1            4             伯克利上 DON'T WALK
         ENT1   2            1
         JMP    FLASH        2             伯克利灯闪
         LDX    AAMBER       8             大道上黄灯
         JOV    *+1          1             消去多余的通过
         LDA    =499994=     3
         JMP    DELAY        4             等候 5 秒
BEGIN    LDX    BGREEN       6             大街上绿灯
         LDA    =1799994=    2
         JMP    DELAY        3             等候至少
         JMP    WAIT         4               18 秒
AGREEN   ALF    CABA                       大道上绿灯
AAMBER   ALF    CBBB                       大道上黄灯
BGREEN   ALF    ACAB                       大街上绿灯
BAMBER   ALF    BCBB                       大街上黄灯
         END    BEGIN                      ▌
```

22.JOSEPHUS PROBLEM

```
N        EQU    24
M        EQU    11
X        ORIG   *+N
0H       ENT1   N-1                1              置每个单元为
         STZ    X+N-1              1                序列中下一个
         ST1    X-1,1              N-1              人的号码
         DEC1   1                  N-1
         J1P    *-2                N-1
         ENTA   1                  1              (现在 rI1=0)
1H       ENT2   M-2                N-1            (假定 M>2)
         LD1    X,1                (M-2)(N-1)     围绕圆圈进行
         DEC2   1                  (M-2)(N-1)       计数
         J2P    *-2                (M-2)(N-1)
         LD2    X,1                N-1            rI1≡幸运的人
```

```
LD3    X,2          N-1    rI2≡不幸运的人
CHAR                N-1    rI3≡下一个人
STX    X,2(4:5)     N-1    存处决数
NUM                 N-1
INCA   1            N-1
ST3    X,1          N-1    从圆圈取人
ENT1   0,3          N-1
CMPA   =N=          N-1
JL     1B           N-1
CHAR                  1    一人剩下
STX    X,1(4:5)       1      他也被痛揍
OUT    X(18)          1    打印答案
HLT                   1
END    0B                  ▌
```

最后一人在位置 15 处。输出之前的总时间是 $(4(N-1)(M+7.5)+16)u$。可能有若干改进,例如,D. Ingalls 建议要有三个字的代码组"DEC2 1;J2P NEXT;JMP OUT",其中 OUT 修改了 NEXT 字段,以便删去一组。渐近地更快的方法请见习题 5.1.1-5。

1.3.3 小节

1. $(1\ 2\ 4)(3\ 6\ 5)$。

2. $a\leftrightarrow c, c\leftrightarrow f; b\leftrightarrow d$。推广到任意排列是显然的。

3. $\begin{pmatrix} a\ b\ c\ d\ e\ f \\ d\ b\ f\ c\ a\ e \end{pmatrix}$。

4. $(a\ d\ c\ f\ e)$。

5. 12。(参见习题 20。)

6. 总的时间对每个紧随"("的空白字,减少 $8u$,因为行 30—32 花费 $4u$ 而行 26—28,33—34,36—38 花费 $12u$。对每个紧随名字的空白字,减少 $2u$,因为行 68—71 花费 $5u$ 而行 42—46 或行 75—79 花费 $7u$。初始的空格和诸循环之间的空格不影响执行的时间。空格的位置对程序 B 毫无影响。

7. $X=2, Y=29, M=5, N=7, U=3, V=1$。由等式(18),总时间为 $2161u$。

8. 是;那样,我们将记住排列的逆,使得 x_i 变成 x_j 当且仅当 $T[j]=i$。(最后的循环形式于是将从右到左地由 T 表构造出来。)

9. 否。例如,给定(6)作为输入,程序 A 将产生"(ADG)(CEB)"作为输出,而程序 B 则将产生"(CEB)(DGA)"。这些答案是等价的,但并不相等,这是由于循环记号的不惟一性所致。对于一个循环所选择的头一个元素,在程序 A 的情况下,是最左边的可用名称,而对于程序 B 的情况,是从右到左所遇到的最后一个可用的不同的名称。

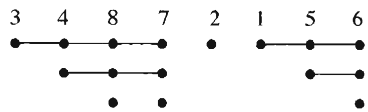
10. (1)基尔霍夫定律推出 $A=1+C-D; B=A+J+P-1; C=B-(P-L); E=D-L; G=E; Q=Z; W=S$。(2)解释:$B$ = 输入的字数 $=16X-1$;C = 非空白字的个数 $=Y$;$D=C-M$;$E=D-M$;F = 查名字表时进行比较的次数;$H=N$;$K=M$;$Q=N$;$R=U$;$S=R-V$;$T=N-V$,由于每一个其它的名称均被标记。(3)加起来,我们有 $(4F+16Y+80X+21N-19M+9U-16V)u$,由于 F 肯定小于 $16NX$,这比程序 A 要稍微好些。在所述情况下的时间为 $983u$,因为 $F=74$。

11. "反射"之;例如,$(a\ c\ f)(b\ d)$ 的逆是 $(d\ b)(f\ c\ a)$。

12. (a)在单元 $L+mn-1$ 中的值,通过转置是固定的,所以我们可以略去不予考虑。否则,如果 $x=n(i-1)+(j-1)<mn-1$,则 $L+x$ 中的值应该变到单元 $L+mx \bmod N=L+(mn(i-1)+m(j-1)) \bmod N=L+m(j-1)+(i-1)$,因为 $mn\equiv 1(\text{modulo } N)$和 $0\le m(j-1)+(i-1)<N$。(b) 如果在每一存储单元中有一位可用(例如,符号位),则我们就能利用像算法 I 那样的算法,随着我们移动诸元素而“标记”它们。[参考 M. F. Berman, *JACM* **5** (1958), 383 ~ 384。]如果没有地方存放一个标记位,这个标记位可以记在一个辅助的表格中,或者,可以使用代表所有非单一元素的循环的表:对于 N 的每个因子 d,由于 m 与 N 互素,我们可以独立地转置那些作为 d 之倍数的元素。含有 x 的循环的长度,当 $\gcd(x,N)=d$ 时,是使得 $m^r\equiv 1(\text{modulo } N/d)$的最小整数 $r>0$。对于每个 d,我们要求 $\varphi(N/d)/r$ 个表示,由这些循环的每一个找一个。某些数论方法可用于这一目的,但它们并不足够简单而真正令人满意。一个有效的但相当复杂的算法可以通过把数论和一个小的标记位表组合在一起得到。[参见 N. Brenner, *CACM* **16** (1973), 692 ~ 694。]最后,有一个类似于算法 J 的方法;它更慢,但不需要辅助存储,而且它在现场实现任何所求的排列。[参见 P. F. Windley, *Comp. J.* **2** (1959), 47 ~ 48; D. E. Knuth, *Proc. IFIP Congress* (1971), **1**, 19 ~ 27; E. G Cate 和 D. W. Twigg, *ACM Tran. Math. Software* **3** (1977), 104 ~ 110; F. E. Fich, J. I. Munro 以及 P. V. Poblete, *SICOMP* **24** (1995), 266 ~ 278。]

13. 用归纳法证明,在步骤 J2 的开始,$X[i]=+j$,当且仅当 $j>m$ 而且在 π 之下 j 变到 i;$X[i]=-j$,当且仅当在 π^{k+1}之下 i 变到 j,其中 k 是使得 π^k 把 i 变成一个小于等于 m 的数的最小非负整数。

14. 以典型循环形式写出已给的排列的逆,并脱去圆括弧,量 $A-N$ 是大于一给定的元素并直接在它右边的连续元素的个数之和。例如,如果原来的排列是(1 6 5)(3 7 8 4),则逆的典型形式是(3 4 8 7)(2)(1 5 6);建立阵列



而且数量 A 是“圆点”的个数,即 16。在第 k 个元素下边的圆点的个数,是在前 k 个元素当中自右到左的极小值的个数(在上面的例子中,在 7 下边有 3 个圆点,因为在 3487 中有 3 个自右到左的极小值)。因此,平均值是 $H_1+H_2+\cdots+H_n=(n+1)H_n-n$。

15. 如果线性表示的第一个字符是 1,则典型表示的最后字符是 1。如果线性表示的第一个字符是 $m>1$,则“$\cdots 1m\cdots$”出现在典型表示中。所以仅有的解是单个对象的排列。(当然,也还有无对象的排列。)

16. 1324, 4231, 3214, 4213, 2143, 3412, 2413, 1243, 3421, 1324, $\cdots$。

17. (a) 循环为 m 循环的概率是 $n!/m$ 除以 $n!H_n$,所以 $p_m=1/(mH_n)$。平均长度是 $p_1+2p_2+3p_3+\cdots=\sum_{m=1}^{n} m/(mH_n)=n/H_n$。(b)由于 m 循环的总数是 $n!/m$,在 m 循环中元素的出现总数是 $n!$。由于对称性,每个元素与其它任何元素同样经常地出现,所以在 m 循环中 k 出现 $n!/n$ 次。因此,在这种情况下,对于所有的 k 和 m,$p_m=1/n$;平均值是$\sum_{m=1}^{n} m/n=(n+1)/2$。

18. 见习题 22(e)。

19. $|P_{n0}-n!/e|=n!/(n+1)!-1/(n+2)!+\cdots$,这是数值递降的交错级数,它小于 $n!/(n+1)!\le\frac{1}{2}$。

20. 每个 m 循环可以以 m 种方式独立地写成;全部有 $\alpha_1+\alpha_2+\cdots$个循环,可以彼此排列在另

一个当中;所以答案是

$$(\alpha_1+\alpha_2+\cdots)!1^{\alpha_1}2^{\alpha_2}3^{\alpha_3}\cdots$$

21. 如果 $n=\alpha_1+2\alpha_2+\cdots$,则为 $1/(\alpha_1!1^{\alpha_1}\alpha_2!2^{\alpha_2}\cdots)$;否则为 0。

证明 在一行中,写出 α_1 个 1 循环,α_2 个 2 循环,等等,连同有空的位置;例如,如果 $\alpha_1=1$, $\alpha_2=2,\alpha_3=\alpha_4=\cdots=0$,则我们将有"(-)(--)(--)"。在所有 $n!$ 种可能的方式下,填充空的位置;我们得到所希望形式的每个排列恰好 $\alpha_1!1^{\alpha_1}\alpha_2!2^{\alpha_2}\cdots$次。

22. a) 如果 $k_1+2k_2+\cdots=n$,则(ii)中的概率为 $\prod_{j\geq0}f(w,j,k_j)$,假定它等于 $(1-w)w^n/k_1!1^{k_1}k_2!2^{k_2}\cdots$;因此

$$\frac{f(w,m,k_m+1)}{f(w,m,k_m)}=\Bigl(\prod_{j\geq0}f(w,j,k_j)\Bigr)^{-1}\prod_{j\geq0}f(w,j,k_j+\delta_{jm})=\frac{w^m}{m(k_m+1)}$$

因此由归纳法

$$f(w,m,k)=\frac{1}{k!}\Bigl(\frac{w^m}{m}\Bigr)^k f(w,m,0)$$

现在条件(i)意味着

$$f(w,m,k)=\frac{1}{k!}\Bigl(\frac{w^m}{m}\Bigr)^k e^{-w^m/m}$$

[换句话说,α_m 是以泊松分布选定的:请见习题 1.2.10-15。]

b) $\sum\limits_{\substack{k_1+2k_2+\cdots=n\\ k_1,k_2,\cdots\geq0}}\bigl(\prod\limits_{j\geq0}f(w,j,k_j)\bigr)=(1-w)w^n\sum\limits_{\substack{k_1+2k_2+\cdots=n\\ k_1,k_2,\cdots\geq0}}P(n;k_1,k_2,\cdots)=(1-w)w^n$。

因此 $\alpha_1+2\alpha_2+\cdots\leq n$ 的概率为 $(1-w)(1+w+\cdots+w^n)=1-w^{n+1}$。

c) ϕ 的平均值为

$$\sum_{n\geq0}\Bigl(\sum_{k_1+2k_2+\cdots=n}\phi(k_1,k_2,\cdots)\Pr(\alpha_1=k_1,\alpha_2=k_2,\cdots)\Bigr)=$$
$$(1-w)\sum_{n\geq0}w^n\Bigl(\sum_{k_1+2k_2+\cdots=n}\phi(k_1,k_2,\cdots)/k_1!1^{k_1}k_2!2^{k_2}\cdots\Bigr)$$

d) 设 $\phi(\alpha_1,\alpha_2,\cdots)=\alpha_2+\alpha_4+\alpha_6+\cdots$。线性组合 ϕ 的平均值是 $\alpha_2,\alpha_4,\alpha_6,\cdots$的平均值之和;$\alpha_m$ 的平均值是

$$\sum_{k\geq0}kf(w,m,k)=\sum_{k\geq1}\frac{1}{(k-1)!}\Bigl(\frac{w^m}{m}\Bigr)e^{-w^m/m}=\frac{w^m}{m}$$

因此 ϕ 的平均值是

$$\frac{w^2}{2}+\frac{w^4}{4}+\frac{w^6}{6}+\cdots=\frac{1-w}{2}(H_1w^2+H_1w^3+H_2w^4+H_2w^5+H_3w^6+\cdots)$$

所求的答案是 $\frac{1}{2}H_{\lfloor n/2\rfloor}$。

e) 置 $\phi(\alpha_1,\alpha_2,\cdots)=z^{\alpha_m}$,并且注意到 ϕ 的平均值是

$$\sum_{k\geq0}f(w,m,k)z^k=\sum_{k\geq0}\frac{1}{k!}\Bigl(\frac{w^m z}{m}\Bigr)^k e^{-w^m/m}=e^{w^m(z-1)/m}=\sum_{j\geq0}\frac{w^{mj}}{j!}\Bigl(\frac{z-1}{m}\Bigr)^j=$$

$$(1-w)\sum_{n\geq 0} w^n\left(\sum_{0\leq j\leq n/m}\frac{1}{j!}\left(\frac{z-1}{m}\right)^j\right)=$$
$$(1-w)\sum_{n\geq 0} w^n G_{nm}(z)$$

因此

$$G_{nm}(z)=\sum_{0\leq j\leq n/m}\frac{1}{j!}\left(\frac{z-1}{m}\right)^j;\ p_{nkm}=\frac{1}{m^k k!}\sum_{0\leq j\leq n/m-k}\frac{(-1/m)^j}{j!}$$

当 $n\geq 2m$ 时，统计是(min 0, ave $1/m$, max $\lfloor n/m\rfloor$, dev $\sqrt{1/m}$)。

23. 常数 λ 是 $\int_0^\infty \exp(-t-E_1(t))\,dt$，其中 $E_1(x)=\int_x^\infty e^{-t}\,dt/t$。见 *Trans. Amer. Soc.* **121** (1966), 340 ~ 357，其中还证明了许多其它结果，特别是最短循环的平均长度近似于 $e^{-\gamma}\ln n$。l_n 之渐近表示的进一步的项已被 Xavier Gourdon 所发现(待发表)；这个级数如下开始：

$$\lambda n+\frac{1}{2}\lambda-\frac{1}{24}e^\gamma n^{-1}+\left(\frac{1}{48}e^\gamma-\frac{1}{8}(-1)^n\right)n^{-2}+$$
$$\left(\frac{17}{3840}e^\gamma+\frac{1}{8}(-1)^n+\frac{1}{6}\omega^{1-n}+\frac{1}{6}\omega^{n-1}\right)n^{-3}$$

其中 $\omega=e^{2\pi i/3}$。William C. Mitchell 已经计算出 λ 的一个高精度的值 = .62432 99885 43550 87099 29363 83100 83724 41796 + [*Math. Comp.* **22** (1968), 411 ~ 415]；λ 与经典的数学常数之间的关系，尚不得知。然而，同一常数在另一个范畴内已由 Karl Dickman 在 *Arkiv för Mat., Astron. och Fys.* **22A**, 10 (1930), 1 ~ 14 中计算出来；直到许多年之后人们才注意到这个巧合[*Theor. Comp. Sci.* **3** (1976), 373。]

24. 见 D. E. Knuth, *Proc. IFIP Congress* (1971), **1**, 19 ~ 27。

25. 一种通过对 N 用归纳法的证明，是以下列事实为基础的：当第 N 个元素为集合中的 s 的成员时，它恰巧为和数贡献

$$\binom{s}{0}-\binom{s}{1}+\binom{s}{2}-\cdots=(1-1)^s=\delta_{s0}$$

另一种通过对 M 用归纳法的证明，是以下列事实为基础的：在 S_M 中但不在 $S_1\cup\cdots\cup S_{M-1}$ 中的元素个数为

$$|S_M|-\sum_{1\leq j<M}|S_j\cap S_M|+\sum_{1\leq j<k<M}|S_j\cap S_k\cap S_M|-\cdots$$

26. 设 $N_0=N$，并设

$$N_k=\sum_{1\leq j_1<\cdots<j_k\leq M}|S_{j_1}\cap\cdots\cap S_{j_k}|$$

则所求公式为

$$N_r-\binom{r+1}{r}N_{r+1}+\binom{r+2}{r}N_{r+2}-\cdots$$

这可根据容斥原理本身来证明，或者通过使用公式

$$\binom{r}{r}\binom{s}{r}-\binom{r+1}{r}\binom{s}{r+1}+\cdots=\binom{s}{r}\binom{s-r}{0}-\binom{s}{r}\binom{s-r}{1}+\cdots=\delta_{sr}$$

如在习题 25 中那样。

27. 设 S_j 为在所述范围内 m_j 的诸倍数，且设 $N = am_1 \cdots m_t$。则 $|S_j \cap S_k| = N/m_j m_k$，等等，所以答案是

$$N - N\sum_{1\le j\le t}\frac{1}{m_j} + N\sum_{1\le j<k\le t}\frac{1}{m_j m_k} - \cdots = N\left(1-\frac{1}{m_1}\right)\cdots\left(1-\frac{1}{m_t}\right)$$

如果我们设 $m_1, \cdots, m_t$ 为整除 N 的素数，则这也解决了习题 1.2.4-30。

29. 当通过一个人时，就给他指定一个新的号数（由 $n+1$ 开始）。于是，第 k 个被处决的人的号数是 $2k$，而且对于号数为 $j, j>n$ 的人，以前的号数为 $(2j) \bmod (2n+1)$。

31. 当 $m=2$ 时，第 k 个被处决的人是 $2n+1-(2n+1-2k)2^{\lfloor \lg(2n/(2n+1-2k))\rfloor}$。[Armin Shams, 2002。]

32. (a)事实上，当 k 为偶数时，$k-1 \le \pi_k \le k+2$；当 k 为奇数时，$k-2 \le \pi_k \le k+1$。(b)从左到右选择指数，并置 $e_k = 1$，当且仅当 k 和 $k+1$ 是在迄今的排列的不同循环之中。[Steven Alpern, *J. Combinatorial Theory*, **B25** (1978), 62 ~ 73。]

33. 对于 $l=0$，令 $(\alpha_{01}, \alpha_{02}; \beta_{01}, \beta_{02}) = (\pi, \rho; \epsilon, \epsilon)$ 以及 $(\alpha_{11}, \alpha_{12}; \beta_{11}, \beta_{12}) = (\epsilon, \epsilon; \pi, \rho)$，其中 $\pi = (1\,4)(2\,3)$，$\rho = (1\,5)(2\,4)$ 及 $\epsilon = (\,)$。

假设对于某个 $l \ge 0$，我们已经作出这样一种构造，其中对于 $0 \le j < m$ 和 $1 \le k \le n$，$\alpha_{jk}^2 = \beta_{jk}^2 = (\,)$。于是排列

$$\begin{aligned}
&(A_{(jm+j')1}, \cdots, A_{(jm+j')(4n)}; B_{(jm+j')1}, \cdots, B_{(jm+j')(4n)}) = \\
&(\sigma^- \alpha_{j1}\sigma, \cdots, \alpha^- \alpha_{jn}\sigma, \tau^- \alpha_{j'1}\tau, \cdots, \tau^- \alpha_{j'n}\tau, \\
&\sigma^- \beta_{jn}\sigma, \cdots, \sigma^- \beta_{j1}\sigma, \tau^- \beta_{j'n}\tau, \cdots, \tau^- \beta_{j'1}\tau; \\
&\sigma^- \beta_{j1}\sigma, \cdots, \sigma^- \beta_{jn}\sigma, \tau^- \beta_{j'1}\tau, \cdots, \tau^- \beta_{j'n}\tau, \\
&\sigma^- \alpha_{jn}\sigma, \cdots, \sigma^- \alpha_{j1}\sigma, \tau^- \alpha_{j'n}\tau, \cdots, \tau^- \alpha_{j'1}\tau)
\end{aligned}$$

有下列性质，即如果 $i=j$ 和 $i'=j'$，则

$$\begin{aligned}
&A_{(im+i')1}B_{(jm+j')1}\cdots A_{(im+i')(4n)}B_{(jm+j')(4n)} = \\
&\sigma^-(1\,2\,3\,4\,5)\sigma\tau^-(1\,2\,3\,4\,5)\tau\sigma^-(5\,4\,3\,2\,1)\sigma\tau^-(5\,4\,3\,2\,1)\tau
\end{aligned}$$

否则乘积为 $(\,)$。当 $im+i' = jm+j'$ 时，选择 $\sigma = (2\,3)(4\,5)$ 和 $\tau = (3\,4\,5)$ 将如所要求那样，作出乘积 $(1\,2\,3\,4\,5)$。

导致从 l 到 $l+1$ 的构造是由 David A. Barrington 给出的[*J. Comp. Syst. Sci* **38** (1989), 150 ~ 164]，他证明了一般的定理，通过这个定理，任何布尔函数都可以表示为 $\{1\,2\,3\,4\,5\}$ 的排列的乘积。通过类似的构造，我们能够，举例说，求出排列 $(\alpha_{j1}, \cdots, \alpha_{jn}; \beta_{j1}, \cdots, \beta_{jn})$ 的顺序，使得对于 $0 \le i, j \le m = 2^{2^l}$，当 $n = 6^{l+1} - 4^{l+1}$ 时，

$$\alpha_{i1}\beta_{j1}\alpha_{i2}\beta_{j2}\cdots\alpha_{in}\beta_{jn} = \begin{cases}(1\,2\,3\,4\,5), & \text{如果 } i<j \\ (\,), & \text{如果 } i \ge j\end{cases}$$

34. 令 $N = m+n$。如果 $m \perp n$，则仅有一个循环，因为对于某个整数 a，每个元素都可以写成 $am \bmod N$ 的形式。而且一般说来，如果 $d = \gcd(m,n)$，则恰有 d 个循环 $C_0, C_1, \cdots, C_{d-1}$，其中 C_j 以某个顺序包含元素 $\{j, j+d, \cdots, j+N-d\}$。为了进行排列，对于 $0 \le j < d$，我们因此可如下进行（如果方便的话，可以并行）：置 $t \leftarrow x_j$ 以及 $k \leftarrow j$；于是当 $(k+m) \bmod N \ne j$ 时，置 $x_k \leftarrow x_{(k+m) \bmod N}$ 和 $k \leftarrow (k+$

m) mod N;最后置 $x_k \leftarrow t$。在这个算法中,关系$(k+m) \bmod N \neq j$ 将成立,当且仅当$(k+m) \bmod N \geq d$,所以我们可以使用任一更有效的测试。[W. Fletcher 和 R. Silver,*CACM* **9** (1966),326。]

35. 令 $M = l + m + n$ 和 $N = l + 2m + n$。对于所要求的重新安排的循环可以通过简单地去掉每个循环中所有大于等于 M 的元素,而从把 k 变成$(k + l + m) \bmod N$ 的$\{0,1,\cdots,N-1\}$的排列的循环得到。(把此行为同习题 29 中的类似情况作比较。)证明:当对于某个 k,且 $k' = (k + l + m) \bmod N$ 和 $k'' = (k' + l + m) \bmod N$ 以及 $k' \geq M$,提示的交换置 $x_k \leftarrow x_{k'}$ 和 $x_{k'} \leftarrow x_{k''}$ 时,我们知道 $x_{k'} = x_{k''}$;因此重新安排 $\alpha\beta\gamma \rightarrow \gamma\beta\alpha$ 以 $x_{k''}$ 代替 x_k。

由此得出,恰有 $d = \gcd(l+m, m+n)$个循环,因此我们可以使用类似于以前的习题的算法。

把这个问题简化为习题 34 中的特殊情况的稍微简单的方法也值得一提,尽管它多做了一些对内存的访问。假设 $\gamma = \gamma'\gamma''$,其中 $|\gamma''| = |\alpha|$。于是我们可以把 $\alpha\beta\gamma'\gamma''$ 变成 $\gamma''\beta\gamma'\alpha$,以及把 $\gamma''\beta$ 同 γ' 交换。如果 $|\alpha| > |\gamma|$,类似的方法也有效。[请见 J. L. Mohammed 和 C. S. Subi, *J. Algorithms* **8** (1987),113 ~ 121。]

1.4.1 小节

1. 调用序列:`JMP MAXN`;或 `JMP MAX100`,如果 $n = 100$。

   入口条件:对于 MAXN 的入口,$rI3 = n$;假定 $n \geq 1$。

   出口条件:与(4)相同。

2.
```
MAX50 STJ    EXIT
      ENT3   50
      JMP    2F
```

3. 入口条件:如果 $rI1 > 0$,则 $n = rI1$;否则 $n = 1$。

   出口条件:rA 和 rI2 如同(4)中那样;rI1 不变;$rI3 = \min(0, rI1)$;rJ = EXIT + 1;如果 $n = 1$,CI 不变,否则根据极大值大于 $X[1]$,等于 $X[1]$且 $rI2 > 1$,或等于 $X[1]$而 $rI2 = 1$,CI 为大于,等于,或小于。

(对于(9)的类似的习题,当然稍微要复杂些。)

4.
```
SMAX1  ENT1  1      r = 1
SMAX   STJ   EXIT   一般的 r
       JMP   2F     像以前一样继续
...
       DEC3  0,1    减去 r
       J3P   1B
EXIT   JMP   *      退出
```

调用序列:`JMP SMAX`;或 `JMP SMAX1` 如果 $r = 1$。

入口条件:$rI3 = n$,假定为正;对于 SMAX 的入口,$rI1 = r$,假定为正。

出口条件:$rA = \max_{0 \leq k < n/r}$ CONTENTS$(X + n - kr)$ = CONTENTS$(X + rI2)$;$rI3 = (n-1) \bmod r + 1 - r = -((-n) \bmod r)$。

5. 可以使用任何其它的寄存器。例如,

调用序列:
```
ENTA  *+2
JMP   MAX100
```

入口条件:无。

出口条件:与(4)中相同。

除了第一条指令变为“MAX100 STA EXIT(0:2)”之外，代码与(1)中相同。

6.(由 Joel Goldberg 和 Roger M. Aarons 给出的解答。)

```
MOVE  STJ    3F
      STA    4F          保存 rA 和 rI2
      ST2    5F(0:2)
      LD2    3F(0:2)     rI2←“NOP A,I(F)”的地址
      LDA    0,2(0:3)    rA←“A,I”
      STA    *+2(0:3)
      LD2    5F(0:2)     恢复 rI2,因为 I 可能为 2。
      ENTA   *           rA←变址地址
      LD2    3F(0:2)
      LD2N   0,2(4:4)    rI2←-F
      J2Z    1F
      DECA   0,2
      STA    2F(0:2)
      DEC1   0,2         rI1←rI1+F
      ST1    6F(0:2)
2H    LDA    *,2
6H    STA    *,2
      INC2   1           增加 rI2 直到它变成零
      J2N    2B
1H    LDA    4F          恢复 rA 和 rI2
5H    ENT2   *
3H    JMP    *           出口到 NOP 指令
4H    CON    0           ▌
```

7.(1) 如果已知程序块是“只读”的，则操作系统可以更有效地分配高速内存。(2) 如果指令不能改动则在硬件中的指令高速缓冲区将是更快和不太昂贵的。(3)和(2)一样，但以流水线代替了“缓存”。如果在进入了流水线之后指令被修改，则流水线需要刷新；必需的校验这个条件的线路是复杂和耗时的。(4)自修改代码不能同时为一个以上进程所使用。(5)自修改代码可以战胜转移跟踪程序(习题 1.4.3.2-7)，它对于“能力测验”(即，对于计算每条指令被执行的次数)是一个重要的诊断工具。

1.4.2 小节

1. 如果一个共行程序仅仅调用另一个一次，则它只不过是子程序；所以我们需要一个这样的应用，其中每个共行程序调用另一个至少有不同的两处。甚至于，通常容易设置某种类型的开关或者使用数据的某种性质，使得当进入到一个共行程序之内的固定的位置时，有可能转到两个所希望的位置之一——因而再次地，将需要的只不过是子程序。当它们之间访问的次数变得更大时，共行程序相应地变得更为有用。

2. 由 IN 找到的第一个字符将失去。[我们首先启动 OUT，因为行 58 ~ 59 为 IN 做必要的初始化。如果我们要首先启动 IN，则我们将要通过比如说“ENT4 - 16”来初始化 OUT，而且如果不知道输出缓冲区是空白的，则要清除之。然后我们可以使行 62 首先跳到行 39。]

3. 几乎是真的，因为 IN 内的“CMPA = 10 = ”就是这程序的唯一比较指令，而且因为“.”的代码是 40。(!)但比较指示器未初始化，而且如果最后的句号的前边是重复的数字，则定不会被注意到。[注：最有效的程序也许是撤消行 40,44 和 48，而且在行 26 与 27 之间插入“CMPA PERIOD”，在行 59 和 60 之间插入“CMPX PERIOD”。于是比较指示器的状态将成为共行程序特征的一部分记录于程序文档中。]

4. 以下是取自于三个具有历史重要性的相当不同的计算机的一些例子：(i)在 IBM 650 上，使用 SOAP 汇编语言，我们将有调用序列“LDD A”和“LDD B”，以及链接“A STD BX AX”和“B STD AX BX”(用两条链接指令于内存中更可取)。(ii)在 IBM 709 上，使用普通的汇编语言，调用序列将是“TSX A,4”或“TSX B,4”；链接指令将如下：

```
A   SXA  BX,4          B   SXA  AX,4
AX  AXT  1 - A1,4      BX  AXT  1 - B1,4
    TRA  1,4               TRA  1,4
```

(iii)在 CDC 1604 上，调用序列将是“返回转移”(SLJ 4)到 A 或 B，而且链接比如说将是

```
A:  SLJ  B1;  ALS  0
B:  SLJ  A1;  SLJ  A
```

在两个连续的 48 位的字中。

5. “STA HOLDAIN; LDA HOLDAOUT”在 OUT 与 OUTX 之间，以及“STA HOLDAOUT; LDA HOLDAIN”在 IN 与 INX 之间。

6. 在 A 之内写“JMP AB”以激活 B，“JMP AC”以激活 C。类似地在 B 和 C 内将使用单元 BA，BC，CA 和 CB。链接是

```
AB  STJ  AX      BC  STJ  BX      CA  STJ  CX
BX  JMP  B1      CX  JMP  C1      AX  JMP  A1
CB  STJ  CX      AC  STJ  AX      BA  STJ  BX
    JMP  BX          JMP  CX          JMP  AX
```

[注：对于 n 个共行程序，将需要 $2(n-1)n$ 个单元以进行这种风格的链接。如果 n 很大，当然可以使用用于链接的“中心化程序”；使用 $3n+2$ 个单元的方法将不难想出。但事实上，上边更快的方法仅需要 $2m$ 个单元，其中 m 是使得共行程序 i 转到共行程序 j 的 (i,j) 偶的个数。当有许多共行程序，每一个都独立地转移到另一个时，控制序列经常是在外部的影响之下，如同 2.2.5 小节所讨论的那样。]

1.4.3.1 小节

1. FCHECK 仅被使用两次，两次都立即跟随对 MEMORY 的调用。所以使 FCHECK 成为对 MEMORY 子程序的一个特殊入口，而且使它把 − R 放入 rI2 中，将会更为有效。

2.
```
SHIFT  J5N   ADDRERROR              ST1   2F(4:5)
       DEC3  5                      J5Z   CYCLE
       J3P   FERROR          2H     SLA   1
       LDA   AREG                   DEC5  1
       LDX   XREG                   J5P   2B
       LD1   1F,3(4:5)              JMP   STOREAX
```

```
         SLA    1                    J1NZ  MEMERROR
         SRA    1                    STZ   SIGN1(0:0)
         SLAX   1                    CMP1  =BEGIN=
         SRAX   1                    JGE   MEMERROR
         SLC    1                    STX   0,1
1H       SRC    1  ▌                 LDA   CLOCK
                                     INCA  2
3.MOVE   J3Z    CYCLE                STA   CLOCK
         JMP    MEMORY               INC1  1
         SRAX   5                    ST1   I1REG
         LD1    I1REG                INC5  1
         LDA    SIGN1                DEC3  1
         JAP    *+3                  JMP   MOVE  ▌
```

4. 只要在行 003 和 004 行之间插入“IN 0(16)”和“JBUS *(16)”指令即可。(当然,在另一台计算机上这将是相当不同的,因为必须转换为 MIX 字符代码。)

5. 中央控制时间为 $34u$,如果需要变址加上 $15u$;GETV 子程序花去 $52u$,如果 $L\neq 0$ 则加上 $5u$;对于 LDA 或 LDX,进行真正的装入的额外时间为 $11u$,对于 LDi 为 $13u$,对于 ENTA 或 ENTX 为 $21u$,对于 ENTi 为 $23u$(如果 $M=0$,则对于后两个时间加上 $2u$)。总加起来,对于 LDA 我们有 $97u$ 的总时间,而对于 ENTA 有 $55u$,对于变址加上 $15u$,而在某些其它的情况下加上 $5u$ 或 $2u$。看起来,在这种情况下的模拟引起大约为 50:1 的速度之比。(含有 $178u$ 模拟时间的测试运行结果,需要 $8422u$ 的实际时间,比例为 47:1。)

7. 执行 IN 或 OUT,把与适当的输入设备相关联的变量置成为希望进行传输的时间。“CYCLE”控制程序在每个循环查询这些变量,看 CLOCK 是否已经超过它们中的每一个(或两个);如是这样,就进行传输而且把变量置为无限大。(当在这种方式下,有多于两个输入输出设备必须加以处理时,则将有许多的变量——因而把它们保存在使用链接内存技术的排了序的表中将更好些;见 2.2.5 小节。)当模拟 HLT 时,需小心地完成输入输出。

8. 错的;如果我们从位置 BEGIN − 1“通过”,rI6 可等于 BEGIN。但那样将出现 MEMERROR,试图 STZ(存零)到 TIME 中! 另一方面,由于行 254,我们总是有 $0\leq rI6\leq$ BEGIN。

1.4.3.2 小节

1. 把行 48 和 49 改变成下面的序列:

```
XREG    ORIG   *+2                    JMP   -1,1
LEAVE   STX    XREG          1H       JMP   *+1
        ST1    XREG+1                 STA   -1,1
        LD1    JREG(0:2)              LD1   XREG+1
        LDA    -1,1                   LDX   XREG
        LDX    1F                     LDA   AREG
        STX    -1,1          LEAVEX   JSJ   *
```

当然,操作符“JSJ”在这里是特别有决定性意义的。

2.
```
* TRACE ROUTINE
         ORIG  *+99
BUF      CON   0
.......... 行 02~04
         ST1   I1REG
.......... 行 05~07
PTR      ENT1  -100
         JBUS  *(0)
         STA   BUF+1,1(0:2)
.......... 行 08~11
         STA   BUF+2,1
.......... 行 12~13
         LDA   AREG
         STA   BUF+3,1
         LDA   I1REG
         STA   BUF+4,1
         ST2   BUF+5,1
         ST3   BUF+6,1
         ST4   BUF+7,1
         ST5   BUF+8,1
         ST6   BUF+9,1
         STX   BUF+10,1
         LDA   JREG(0:2)
         STA   BUF+1,1(4:5)
         ENTA  8
         JNOV  1F
         ADD   BIG
1H       JL    1F
         INCA  1
         JE    1F
         INCA  1
1H       STA   BUF+1,1(3:3)
         INC1  10
         J1N   1F
         OUT   BUF-99(0)
         ENT1  -100
1H       ST1   PTR(0:2)
.......... 行 14~31
         LD1   I1REG
.......... 行 32~35
         ST1   I1REG
.......... 行 36~48
         LD1   I1REG
.......... 行 49~50
B4       EQU   1(1:1)
BIG      CON   B4-8,B4-1(1:1)   ▌
```

在执行了所有的跟踪之后,应该调用输出最后的缓冲区并将 0 号磁带设备重绕的辅助程序。

3. 磁带更快;而且在跟踪时把这个信息编辑成字符,将消耗太多的空间。而且磁带的内容可有选择地打印。

4. 将得不到习题 6 中所希望的真正跟踪,因为违反了正文中提到的限制(a)。跟踪 CYCLE 的第一次尝试,将引起回到跟踪 ENTER+1 的循环,因为 PREC 被压垮了。

6. 建议:保持一份已经为外部程序改变了的跟踪区域内每个内存单元之值的表格。

7. 这个例程将扫描程序,直至找到第一条转移指令(或条件转移指令);在修改了该指令以及紧接着的指令之后,它将恢复寄存器并允许程序一口气执行所有的指令,直到该点为止。[如果程序修改它自己的转移指令,这项技术可能失败。为了实用的目的,我们可以把这样一种做法抛开,只是 STJ 除外,我们大概应该对它独立处理。]

1.4.4 小节

1. (a) 否,输入操作可能尚未完成。(b) 否,输入操作可能只快一点点,而这太冒险了。

2.
```
ENT1   2000
JBUS   *(6)
MOVE   1000(50)
MOVE   1050(50)
OUT    2000(6)   ▌
```

3.
```
WORDOUT  STJ   1F                      DEC5   100
         STA   0,5                     JMP    2B
         INC5  1           BUFMAX      CON    ENDBUF1
2H       CMP5  BUFMAX      * BUFFER    AREAS
1H       JNE   *           OUTBUF1     ORIG   * +100
         OUT   -100,5(V)   ENDBUF1     CON    ENDBUF2
         LD5   0,5         OUTBUF2     ORIG   * +100
         ST5   BUFMAX      ENDBUF2     CON    ENDBUF1   ▌
```

在这程序的开始处，给出指令“ENT5 OUTBUF1”。在程序的末尾，可以是

```
LDA    BUFMAX          INC5    1
DECA   100,5           CMP5    BUFMAX
JAZ    * + 6           JNE     * - 3
STZ    0,5             OUT     - 100,5(V)
```

4. 如果计算时间恰巧等于输入输出时间（这是最有利的情况），则计算机与外部设备两者同时运行将花费当它们独自运行时的一半的时间。形式地说，设 C 是整个程序的运行时间，设 T 是所需总的输入输出时间；则通过缓冲的最好可能运行时间是 $\max(C,T)$，而没有缓冲的运行时间是 $C+T$；当然 $\frac{1}{2}(C+T)\le\max(C,T)\le C+T$。

然而，有些设备，它有一个“关闭惩罚”，即如果对于该设备的两次访问之间，出现一个太长的时间间隔，则会导致损失额外的时间；在这样一种情况下，可能有比 2:1 更好的比例。（例如，见习题 19。）

5. 最好的比例为 $(n+1):1$。

6. $\left\{\begin{matrix}\texttt{IN} & \texttt{INBUF1(U)}\\ \texttt{ENT6} & \texttt{INBUF2+99}\end{matrix}\right\}$ 或 $\left\{\begin{matrix}\texttt{IN} & \texttt{INBUF2(U)}\\ \texttt{ENT6} & \texttt{INBUF1+99}\end{matrix}\right\}$

（可能仅仅在必要的情况下，前面加上 IOC 0(U)以重绕带。）

7. 一种方法是使用共行程序

```
INBUF1    ORIG   *+100                         INC6   1
INBUF2    ORIG   *+100                         J6N    2B
1H        LDA    INBUF2+100,6                  IN     INBUF1(U)
          JMP    MAIN                          ENN6   100
          INC6   1                             JMP    1B
          J6N    1B               WORDIN       STJ    MAINX
WORDIN1   IN     INBUF2(U)        WORDINX      JMP    WORDIN1
          ENN6   100              MAIN         STJ    WORDINX
2H        LDA    INBUF1+100,6     MAINX        JMP    *      ▌
          JMP    MAIN
```

加上几条利用特殊情况的指令，将使得这个程序实际上比(4)更快。

8. 在图 23 所示的时间里，两个红的缓冲区已经用行的映像加以填充，而且正在打印的是由 NEXTR 指出的一个。同时，程序正在 RELEASE 与 ASSIGN 之间进行计算。当程序进行 ASSIGN

(指定)时，由 NEXTG 指示的绿色缓冲区变成黄色；NEXTG 顺时针移动，而且程序开始填黄色缓冲区。当完成输出操作时，NEXTR 顺时针移动，刚刚被打印的缓冲区转为绿色，而且剩下的红色缓冲区被打印。最后，这个程序 RELEASE(释放)黄色缓冲区而且它也为随后的打印做好准备。

9,10,11.

时间	动作($N=1$)	动作($N=2$)	动作($N=4$)
0	ASSIGN(BUF1)	ASSIGN(BUF1)	ASSIGN(BUF1)
1000	RELEASE,OUT BUF1	RELEASE,OUT BUF1	RELEASE,OUT BUF1
2000	ASSING(等候)	ASSIGN(BUF2)	ASSIGN(BUF2)
3000		RELEASE	RELEASE
4000		ASSIGN(等候)	ASSIGN(BUF3)
5000			RELEASE
6000			ASSIGN(BUF4)
7000			RELEASE
8000			ASSIGN(等候)
8500	指定 BUF1,输出停止	指定 BUF1,OUT BUF2	指定 BUF1,OUT BUF2
9500	RELEASE,OUT BOF1	RELEASE	
10500	ASSIGN(等候)	ASSIGN(等候)	
15500			RELEASE

等等。当 $N=1$ 时，总的时间是 $110000u$；当 $N=2$ 时，它是 $89000u$；当 $N=3$ 时，它是 $81500u$；而且当 $N\geq4$ 时，它是 $76000u$。

12. 用下列代码代替程序 B 的最后三行：

```
    STA   2F
    LDA   3F
    CMPA  15,5(5:5)
    LDA   2F
    LDA   -1,5
    DEC6  1
    JNE   1B
    JMP   COMPUTE
    JMP   *-1(或 JMP COMPUTEX)
2H  CON   0
3H  ALF   ␣␣␣␣. ▌
```

13.
```
    JRED  CONTROL(U)
    J6NZ  *-1        ▌
```

14. 如果 $N=1$，则算法失败(当 I/O 在进行时，可能访问了缓冲区)；否则，这个构造将产生有两个黄色缓冲区的效果。如果计算程序一次要访问两个缓冲区，则这是有用的，虽然它冻结缓冲区的空间。一般地说，对于 RELEASE(释放)的过多 ASSIGN(指定)将是非负的，而且不大于 N。

15.
```
U        EQU    0
V        EQU    1
BUF1     ORIG   *+100
BUF2     ORIG   *+100
BUF3     ORIG   *+100
TAPECPY  IN     BUF1(V)
```

```
              ENT1   99              OUT    BUF3(V)
        1H    IN     BUF2(U)         DEC1   3
              OUT    BUF1(V)         J1P    1B
              IN     BUF3(U)         OUT    BUF1(V)
              OUT    BUF2(V)         HLT
              IN     BUF1(U)         END    TAPECPY  ▌
```

这是在图 26 中所指出的算法的特殊情况。

18. 部分解答:在下列算法中,t 是变量,当 I/O 设备活动时其值为 0,而当 I/O 设备空闲时等于 1。

算法 A(ASSIGN,通常状态的子程序)

这个算法与算法 1.4.4A 相比没有什么变化。

算法 R(RELEASE,通常状态的子程序)

R1. n 增加 1。

R2. 如果 $t=0$,则引起一个中断,转到步骤 B3(使用 INT 操作符)。 ▌

算法 B(缓冲区控制程序,它处理中断)

B1. 重新启动主程序。

B2. 如果 $n=0$,则置 $t \leftarrow 0$ 并转到步骤 B1。

B3. 置 $t \leftarrow 1$ 并从由 NEXTR 所确定的缓冲区初始化 I/O。

B4. 重新启动主程序;一个"中断完成"条件将中断它并导致步骤 B5。

B5. 推进 NEXTR 到下一个顺时针的缓冲区。

B6. n 减 1,并转到步骤 B2。 ▌

19. 如果 $C \leqslant L$ 我们可以有 $t_k=(k-1)L$, $u_k=t_k+T$,而且 $v_k=u_k+C$,当且仅当 $NL \geqslant T+C$。如果 $C>L$ 则情况就更复杂;我们可以有 $u_k=(k-1)C+T$ 和 $v_k=kC+T$,当且仅当有整数 $a_1 \leqslant a_2 \leqslant \cdots \leqslant a_n$ 使得对于 $N<k \leqslant n$, $t_k=(k-1)L+a_kP$ 满足 $u_k-T \geqslant t_k \geqslant v_{k-N}$。等价的条件是对于 $N<k \leqslant n$, $NC \geqslant b_k$,其中 $b_k=C+T+((k-1)(C-L)) \bmod P$。令 $c_l=\max\{b_{l+1},\cdots,b_n,0\}$;则当 l 增加时,c_l 减小,而使进程保持稳定地运行的最小的 N 值是使 $c_l/l \leqslant C$ 的最小的 l。因为 $c_l<C+T+P$ 和 $c_l \leqslant L+T+n(C-L)$,N 的这个值绝不超过 $\lceil \min\{C+T+P, L+T+n(C-L)\}/C \rceil$。[请见 A. Itai 和 Y. Raz, *CACM* **31** (1988), 1338~1342。]

在所述的例子中,我们因此有(a) $N=1$; (b) $N=2$; (c) $N=3$, $c_N=2.5$; (d) $N=35$, $c_N=51.5$; (e) $N=51$, $c_N=101.5$; (f) $N=41$, $c_N=102$; (g) $N=11$, $c_N=109.5$; (h) $N=3$, $c_N=149.5$; (i) $N=2$, $c_N=298.5$。

2.1 节

1.(a)SUIT(NEXT(TOP)) = SUIT(NEXT(242)) = SUIT(386) = 4。(b)Λ。

2. 每当 V 是一个链接变量时(否则 CONTENTS(V)就没有意义),它的值就不是 Λ。在像这里的情况下,避免使用 LOC 是明智的。

3. 置 NEWCARD←TOP,而且如果 TOP≠Λ,则置 TOP←NEXT(TOP)。

4. **C1**. 置 X←LOC(TOP)。(为方便起见,我们做一个合理的假定:TOP≡NEXT(LOC(TOP)),即是,TOP 的值出现于保存它的单元的 NEXT 字段。这个假定与程序(5)是相容的,而且它使我们免去编写对于空叠情况的专用程序。)

C2. 如果 NEXT(X)≠Λ,则置 X←NEXT(X)并重复这一步骤。

C3. 置 NEXT(X)←NEWCARD,NEXT(NEWCARD)←Λ,TAG(NEWCARD)←1。 ▌

5. **D1**. 置 X←LOC(TOP),Y←TOP。(见上面的步骤 C1。由假设,Y≠Λ。在下面的算法中,在 Y = NEXT(X)的意义下,X 滞后 Y 一步。)

D2. 如果 NEXT(Y)≠Λ ,则置 X←Y,Y←NEXT(Y),并重复这一步骤。

D3. (现在 NEXT(Y) = Λ,所以 Y 指向底下的卡片;X 指向次末一张卡片。)置 NEXT(X)←Λ,NEWCARD←Y。 ▌

6. (b)和(d)的记号。不是(a)! CARD 是一个节点,不是指向节点的链接。

7. 序列 a)给出 NEXT(LOC(TOP)),在这种情况下它恒等于 TOP 的值;序列 b)是正确的。没必要弄混;当 X 是一个数值变量时,考虑类似的例子:为把 X 写入寄存器 A,我们写 LDA X,而不写 ENTA X,因为后者把 LOC(X)写入寄存器中。

8. 设 rA≡N,rI1≡X。

```
ENTA    0             B1. N←0
LD1     TOP           X←TOP
J1Z     *+4           B2. X = Λ 吗?
INCA    1             B3. N←N+1
LD1     0,1(NEXT)     X←NEXT(X)
J1NZ    *-2                                  ▌
```

9. 设 rI2≡X。

```
PRINTER   EQU    18              打印机的设备号
TAG       EQU    1:1
NEXT      EQU    4:5             字段的定义
NAME      EQU    0:5
PBUF      ALF    PILE            在空叠的情况下
          ALF    EMPTY             打印的信息
          ORIG   PBUF+24
BEGIN     LD2    TOP             置 X←TOP
          J2Z    2F              是空叠吗?
1H        LDA    0,2(TAG)        rA←TAG(X)
          ENT1   PBUF            对 MOVE 指令准备好
          JBUS   *(PRINTER)      等候打印机准备好
          JAZ    *+3             TAG = 0(卡片面朝上)吗?
          MOVE   PAREN(3)        否:复写圆括弧
          JMP    *+2
          MOVE   BLANKS(3)       是:复写空格
          LDA    1,2(NAME)       rA←NAME(X)
          STA    PBUF+1
          LD2    0,2(NEXT)       置 X←NEXT(X)
2H        OUT    PBUF(PRINTER)   打印此行
          J2NZ   1B              如果 X≠Λ,则重复打印循环
DONE      HLT
PAREN     ALF    (
```

```
BLANKS      ALF
            ALF )
            ALF
```

2.2.1 小节

1. 是。(在两端之一一致地插入所有项。)

2. 为得到 325641,作 SSSXXSSXSXXX(在下列习题的记号下)。不可能达到 154623 的次序,因为仅仅在 3 插入栈之前,从栈中撤消 2 时,2 才能在 3 之前。

3. 可允许的序列是当我们从左到右来读时,其中 X 的个数绝不超过 S 的个数这样的序列。

两个不同的可允许的序列必须给出不同的结果,因为如果两个序列相一致到这样一个位置,即在该处一个序列为 S 而另一个为 X,则后一个序列输出一个符号,这个符号,不可能是在由前一个序列的 S 刚刚插入的符号之前的输出。

4. 这个问题等价于许多其它有趣的问题,诸如二叉树的枚举,把圆括弧插入公式中的方法数,以及把多边形分成三角形的方法数,而且它早在 1759 年就出现于欧拉和 Von Segner 的短文中(见 2.3.4.6 小节)。

下面的精彩解法使用由 D. André 给出的"反射原理":显然有$\binom{2n}{n}$个含 S 和 X 各 n 个的序列。剩下的是计算不允许的序列数(它包含正确个数的 S 和 X,但是违背其它条件)。在任何不允许的序列中,定出使得 X 的个数超过 S 的个数的第一个 X 的位置。然后,在导致并包括这个 X 的部分序列中,以 S 代替所有的 X 并以 X 代替所有的 S。结果是一个有$(n+1)$个 S 和$(n-1)$个 X 的序列。反过来,对于后一种类型的每个序列,我们都能逆转这个过程,而且找出导致它的前一种类型的不允许序列。例如,序列 XXSXSSSXXSSS 必然来自 SSXSXXXXXSSS。这个对应说明,不允许序列的个数是$\binom{2n}{n-1}$。因此 $a_n=\binom{2n}{n}-\binom{2n}{n-1}$。[*Comptes Rendus Acad. Sci.* **105** (Paris, 1887), 436~437。]

利用同样的思想,我们可以解决概率论的更一般的"抽签问题",它实际上是要枚举所有的具有给定个数的 S 和 X 的部分可允许序列,这个问题实际上早在 1708 年就由 Abraham de Moivre 解决了,他证明了,包含 l 个 A 和 m 个 B,以及包含至少一个 A 的个数比 B 的个数多 n 的初始子串的序列的个数是 $f(l,m,n)=\binom{l+m}{\min(m,l-n)}$。特别是,如上面所述,$a_n=\binom{2n}{n}-f(n,n,1)$。(棣莫弗未加证明地指出这个结果[*Philos. Trans.* **27** (1711), 262~263];但从他的论文的其它章节看,显然他知道怎么来证明它,因为当 $l\geqslant m+n$ 时公式显然为真,而且因为对于类似问题他的生成函数方法,通过简单的代数运算就可产生对称条件 $f(l,m,n)=f(m+n,l-n,n)$。)对于抽签问题的后续历史及其某些推广,见 D. E. Barton 和 C. L. Mallows 的全面评述,*Annals of Math. Statistics* **36** (1965), 236~260;也见习题 2.3.4.4-32 及 5.1.4 小节。

在这里,我们给出一个新的通过使用双重生成函数的方法,来解决抽签问题,因为这个方法得以解决诸如习题 11 那样更困难的问题。

设 g_{nm} 是长度为 n 的 S 和 X 的序列的个数,在这些序列中,如果我们从左边计数,则 X 的个数绝不超过 S 的个数,而且总共是 S 比 X 多 m 个。于是 $a_n=g_{(2n)0}$。显然 g_{nm} 为 0,除非 $m+n$ 为偶数。容易看出这些数据可由递推关系

$$g_{(n+1)m}=g_{n(m-1)}+g_{n(m+1)},\ m\geqslant 0,\ n\geqslant 0;\ g_{0m}=\delta_{0m}$$

确定。考虑双重生成函数 $G(x,z)=\sum_{n,m}g_{nm}x^m z^n$,而且令 $g(z)=G(0,z)$。上边的递推关系与

方程$\left(x+\frac{1}{x}\right)G(x,z)=\frac{1}{x}g(z)+\frac{1}{z}(G(x,z)-1)$，即 $G(x,z)=\frac{zg(z)-x}{z(x^2+1)-x}$等价。如果我们置 $x=0$，则这个方程不幸地什么也没有告诉我们，但是，我们可把分母分解成 $z(1-r_1(z)x)(1-r_2(z)x)$，其中

$$r_1(z)=\frac{1}{2z}(1+\sqrt{1-4z^2}),\quad r_2(z)=\frac{1}{2z}(1-\sqrt{1-4z^2})$$

(注意，$r_1+r_2=1/z$；$r_1r_2=1$。)我们现在以启发式进行；问题是要求 $g(z)$的某个值，使得由上边的公式给出的 $G(x,z)$，有一个 x 和 z 的无穷幂级数展开。函数 $r_2(z)$有一个幂级数，而 $r_2(0)=0$；而且对于固定的 z，值 $x=r_2(z)$使得 $G(x,z)$之分母为0。这就提示我们，可以选择 $g(z)$使得当 $x=r_2(z)$时分子亦为0；换句话说，我们大概应取 $zg(z)=r_2(z)$。以这个选择，$G(x,z)$的等式简化为

$$G(x,z)=\frac{r_2(z)}{z(1-r_2(z)x)}=\sum_{n\geq 0}(r_2(z))^{n+1}x^nz^{-1}$$

这是一个满足原来等式的幂级数展开，所以我们必然已找到 $g(z)$的正确选择。

$g(z)$的系数就是对我们问题的解答。实际上，我们能进一步进行而且对于 $G(x,z)$的所有系数能推导出简单的形式：由二项式定理，

$$r_2(z)=\sum_{k\geq 0}z^{2k+1}\binom{2k+1}{k}\frac{1}{2k+1}$$

设 $w=z^2$ 和 $r_2(z)=zf(w)$。则在习题 1.2.6-25 的记号下 $f(w)=\sum_{k\geq 0}A_k(1,-2)w^k$；因此

$$f(w)^r=\sum_{k\geq 0}A_k(r,-2)w^k$$

我们现在有

$$G(x,z)=\sum_{n,m}A_m(n+1,-2)x^nz^{2m+n}$$

所以通解为

$$g_{(2n)(2m)}=\binom{2n+1}{n-m}\frac{2m+1}{2n+1}=\binom{2n}{n-m}-\binom{2n}{n-m-1}$$

$$g_{(2n+1)(2m+1)}=\binom{2n+2}{n-m}\frac{2m+2}{2n+2}=\binom{2n+1}{n-m}-\binom{2n+1}{n-m-1}$$

5. 如果 $j<k$ 和 $p_j<p_k$，则我们必须在 p_k 置入栈之前，把 p_j 取走；如果 $p_j>p_k$，则我们必须把 p_k 留在栈上，直到 p_j 放到栈之后。把这两条规则结合起来，条件 $i<j<k$ 和 $p_j<p_k<p_i$ 是不可能的，因为它意味着 p_j 必须在 p_k 之前和在 p_i 之后离开，而 p_i 又出现在 p_k 之后。

反之，通过使用算法"对于 $j=1,2,\cdots,n$，输入 0 个或多个项(需要多少个就输入多少个)，直到 p_j 第一次出现在栈中为止；然后输出 p_j"，可以得到所要的排列。仅当我们达到某个 j，p_j 不在栈的顶上但它被 $k>j$ 的某个元素 p_k 所覆盖时，这个算法才可能失败。由于这个栈上的数值总在单调增加，我们有 $p_j<p_k$。元素 p_k 可能已到达那里，因为对于某个 $i<j$，它小于 p_i。

P. V. Ramanan [*SICOMP* **13** (1984), 167 ~ 169]已经说明，当除了栈之外，还可以自由地使用 m 个辅助的存储单元时，如何表征可得到的排列(这个问题的推广令人惊讶地困难)。

6. 按队列的性质，仅平凡的一个，1 2 ⋯ n。

7. 首先输出 n 的输入受限的双端队列,必须以它的头 n 个操作那样的顺序,简单地把值 1,2,…,n 放到双端队列上。首先输出 n 的输出受限的双端队列,必须把值 $p_1p_2\cdots p_n$ 放到它的双端队列上,如同它的前 n 个操作那样。因此我们求出惟一的答案(a) 4 1 3 2,(b) 4 2 1 3,(c) 4 2 3 1。

8. 当 $n\leqslant 4$ 时,无;当 $n=5$ 时,有 4 个(见习题 13)。

9. 由向后的操作,我们可以通过输出受限的双端队列,得到任何输入受限的排列反演之逆的反演,且反之亦然。这个规则就在两组排列之间建立一一对应关系。

10.(i)应当有 n 个 X 和 n 个组合的 S 和 Q。(ii)如果我们从左边读的话,X 的个数绝不超过组合的 S 和 Q 的个数。(iii)每当 X 的个数等于组合的 S 和 Q 的个数时(从左边读),下一个字符必须是 Q。(iv)两个操作 XQ 永远不得以这个次序相邻。

显然,规则(i)和(ii)是必要的。加上额外的规则(iii)和(iv)以消除二义性,因为当双端队列为空时 S 就等同于 Q,而且 XQ 总可为 QX 所代替。于是,任何可得到的序列至少对应于一个可允许的序列。

为了说明两个可允许的序列给出不同的排列,考虑相同到某一个点的一些序列,于是一个序列有一个 S,另一个有一个 X 或 Q。由于按(iii),双端队列非空,显然通过两个序列得到不同的排列(相对于通过 S 插入元素的次序)。其余情况是序列 A,B 符合到某一个点,而后序列 A 有 Q,序列 B 有 X。序列 B 在该处可能有更多的 X,而按(iv)它们必须接以 S,所以这两个排列还是不同的。

11. 如在习题 4 中那样进行,我们设 g_{nm} 是长度为 n 的部分可允许序列的个数,它在双端队列上保留 m 个元素,不以符号 X 结束;h_{nm} 类似地定义,是那些以 X 结束的序列。我们有 $g_{(n+1)m}=2g_{n(m-1)}+h_{n(m-1)}[m>1]$,$h_{(n+1)m}=g_{n(m+1)}+h_{n(m+1)}$。类似于习题 4 中的定义,定义 $G(x,z)$ 和 $H(x,z)$;我们有

$$G(x,z)=xz+2x^2z^2+4x^3z^3+(8x^4+2x^2)z^4+(16x^5+8x^3)z^5+\cdots$$

$$H(x,z)=z^2+2xz^3+(4x^2+2)z^4+(8x^3+6x)z^5+\cdots$$

置 $h(z)=H(0,z)$,我们求得 $z^{-1}G(x,z)=2xG(x,z)+x(H(x,z)-h(z))+x$,且 $z^{-1}H(x,z)=x^{-1}G(x,z)+x^{-1}(H(x,z)-h(z))$;因此

$$G(x,z)=\frac{xz(x-z-xh(z))}{x-z-2x^2z+xz^2}$$

如同在习题 4 中那样,我们试图选择 $h(z)$ 以使分子与分母的因子相消。我们发现 $G(x,z)=xz/(1-2xr_2(z))$,其中

$$r_2(z)=\frac{1}{4z}(z^2+1-\sqrt{(z^2+1)^2-8z^2})$$

利用 $b_0=1$ 的约定,所求的生成函数成为

$$\frac{1}{2}(3-z-\sqrt{1-6z+z^2})=1+z+2z^2+6z^3+22z^4+90z^5+\cdots$$

通过微分,我们找出便于计算的递推关系:$nb_n=3(2n-3)b_{n-1}-(n-3)b_{n-2}$,$n\geqslant 2$。

另一个解决这个问题的方法是由 V. Pratt 提出的,该方法使用对于串的集合的上下文无关文法(参照第 10 章)。对于所有 $n\geqslant 0$,具有产生式 $S\to q^n(Bx)^n$,$B\to sq^n(Bx)^{n+1}B$,以及 $B\to\epsilon$ 的无限文法,是无二义性的,而且它允许我们计算具有 n 个 x 的串的个数,如同在习题 2.3.4.4-31 中那样。

12. 由斯特林公式,我们有 $a_n=4^n/\sqrt{\pi n^3}+O(4^n n^{-5/2})$。为分析 b_n,首先让我们考虑当

$|\alpha|<1$ 时估计在$\sqrt{1-w}\sqrt{1-\alpha w}$的幂级数中 w^n 的系数的一般问题。对充分小的 α,我们有

$$\sqrt{1-w}\sqrt{1-\alpha w}=\sqrt{1-w}\sqrt{1-\alpha+\alpha(1-w)}=\sqrt{1-\alpha}\sum_k\binom{1/2}{k}\beta^k(1-w)^{k+1/2}$$

其中 $\beta=\alpha/(1-\alpha)$;因此所求系数是$(-1)^n\sqrt{1-\alpha}\sum_k\binom{1/2}{k}\beta^k\binom{k+1/2}{n}$。现在

$$(-1)^n\binom{k+1/2}{n}=\binom{n-k-3/2}{n}=\frac{\Gamma(n-k-1/2)}{\Gamma(n+1)\Gamma(-k-1/2)}=-\frac{(-1/2)^{\overline{k+1}}}{\sqrt{\pi}n}n^{\overline{-k-1/2}}$$

而且由等式 1.2.11.1-(16),$n^{\overline{-k-1/2}}=\sum_{j=0}^m\left[\begin{matrix}-k-1/2\\-k-1/2-j\end{matrix}\right]n^{-k-1/2-j}+O(n^{-k-3/2-m})$。于是,我们得到渐近级数$[w^n]\sqrt{1-w}\sqrt{1-\alpha w}=c_0n^{-3/2}+c_1n^{-5/2}+\cdots+c_mn^{-m-3/2}+O(n^{-m-5/2})$,其中

$$c_j=\sqrt{\frac{1-\alpha}{\pi}}\sum_{k=0}^{j}\binom{1/2}{k}(-1/2)^{\overline{k+1}}\left\{\begin{matrix}j+1/2\\k+1/2\end{matrix}\right\}\frac{\alpha^k}{(1-\alpha)^k}$$

对于 b_n,我们写 $1-6z+z^2=(1-(3+\sqrt{8})z)(1-(3-\sqrt{8})z)$,并令 $w=(3+\sqrt{8})z$,$\alpha=(3-\sqrt{8})/(3+\sqrt{8})$,得到渐近公式

$$b_n=\frac{(\sqrt{2}-1)(3+\sqrt{8})^n}{2^{3/4}\pi^{1/2}n^{3/2}}(1+O(n^{-1}))=\frac{(\sqrt{2}+1)^{2n-1}}{2^{3/4}\pi^{1/2}n^{3/2}}(1+O(n^{-1}))$$

13. V. Pratt 发现,一个排列是不能得到的,当且仅当,对于某个 $k\geq1$,它包含相对数量分别为

$$5,2,7,4,\cdots,4k+1,4k-2,3,4k,1 \text{ 或 } 5,2,7,4,\cdots,4k+3,4k,1,4k+2,3$$

的子序列,或者是同一序列但交换其后两个元素,或者交换其 1 和 2,或者两者兼有所得的子序列。因此,对于 $k=1$,禁止的样式为 52341,52314,51342,51324,5274163,5274136,5174263,5174236。[*STOC* **5** (1973),268~277。]

14. (R. Melville 提供的解,1980) 设 R 和 S 是栈,它们使得队列从 R 的顶运行到它的底,接着从 S 的底到它的顶。当 R 为空时,把 S 的元素弹出到 R 上直到 S 变空为止。为了从前端删去,弹出 R 的顶,R 将不是空的,除非整个队列都为空。为插入到后部,压入 S 中(除非 R 为空)。在离开队列之前,每个元素至多被压入两次,至多也被弹出两次。

2.2.2 小节

1. M-1(不是 M)。如果我们允许 M 个项,像(6)和(7)那样,则将不可能通过检查 R 和 F 来区别全满的队列与空的队列,因为仅有 M 种可能性能被检测。放弃一个存储单元,比起去编写过于复杂的程序来,要更好些。

2. 从尾部删去:如果 R = F,则 UNDERFLOW;Y←X[R];如果 R = 1,则 R←M,否则 R←R - 1。在前头插入:置 X[F]←Y;如果 F = 1 则 F←M,否则 F←F - 1;如果 F = R 则 OVERFLOW。

3. a) LD1 I;LDA BASE,7:1。这花去 5 个时钟周期,而不是像在(8)中那样的 4 个或 8 个。

b) *解法 1*:LDA BASE,2:7,其中每个基地址通过 $I_1=0$,$I_2=1$ 来存储。*解法 2*:如果希望以 $I_1=I_2=0$ 来存基地址,则我们可以写 LDA X2,7:1,其中单元 X2 含有 NOP BASE,2:7。第二个解法多花去一个时钟周期,但是允许通过任何变址寄存器来使用基表。

c)这等价于“LD4 X(0:2)”,而且花去同样的执行时间,但当 X(0:2)包含 -0 时 rI4 将置成 +0。

4. a) NOP *,7。b) LDA X,7:7(0:2)。c) 这是不可能的;代码 LDA Y,7:7,其中单元 Y 含有

NOP X,7:7,破坏了对 7:7 的限制(见习题 5)。d) LDA X,7:1 连同辅助常数

```
X    NOP    *+1,7:2
     NOP    *+1,7:3
     NOP    *+1,7:4
     NOP    0,5:6
```

执行时间是 6 个单位。e) INC6 X,7:6,其中 X 含有 NOP 0,6:6。

5.a)考虑指令 ENTA 1000,7:7 连同内存配置

单元	ADDRESS	I_1	I_2
1000:	1001	7	7
1001:	1004	7	1
1002:	1002	2	2
1003:	1001	1	1
1004:	1005	1	7
1005:	1006	1	7
1006:	1008	7	7
1007:	1002	7	1
1008:	1003	7	2

以及 rI1 = 1, rI2 = 2。我们求得 1000,7,7 = 1001,7,7,7 = 1004,7,1,7,7 = 1005,1,7,1,7,7 = 1006,7,1,7,7 = 1008,7,7,1,7,7 = 1003,7,2,7,1,7,7 = 1001,1,1,2,7,1,7,7 = 1002,1,2,7,1,7,7 = 1003,2,7,1,7,7 = 1005,7,1,7,7 = 1006,1,7,1,7,7 = 1007,7,1,7,7 = 1002,7,1,1,7,7 = 1002,2,2,1,1,7,7 = 1004,2,1,1,7,7 = 1006,1,1,7,7 = 1007,1,7,7 = 1008,7,7 = 1003,7,2,7 = 1001,1,1,2,7 = 1002,1,2,7 = 1003,2,7 = 1005,7 = 1006,1,7 = 1007,7 = 1002,7,1 = 1002,2,2,1 = 1004,2,1 = 1006,1 = 1007。(手工来做这个推导的一个更快的方法,将是逐次地计算在单元 1002,1003,1007,1008,1005,1006,1004,1001,1000 中以此顺序确定的地址,但是看起来,计算机实质上将需要像上面这样进行求值。)作者试验过若干奇特的方案,以便在计算地址的同时改变内存内容,而且已经设计成使得在得到了最后的地址时再次恢复每一事项。类似的算法出现于 2.3.5 小节中。然而,这些企图并没有获得收获,而且看起来恰恰是没有足够的单元来存储必要的信息。

b),c) 设 H 和 C 是辅助寄存器,而且设 N 是一个计数器。对于单元 L 中的指令,为了得到有效的地址 M,按如下方法来做:

A1. [初始化] 置 H←0,C←L,N←0。(在这个算法中,C 将是"当前"单元,H 用来把各个变址寄存器的内容加到一起,而 N 测量间接寻址的"深度")。

A2. [考察地址] 置 M←ADDRESS(C)。如果 $I_1(C) = j$, $1 \le j \le 6$,则置 M←M + rIj。如果 $I_2(C) = j$, $1 \le j \le 6$,则置 H←H + rIj。如果 $I_1(C) = I_2(C) = 7$,则置 N←N + 1, H←0。

A3. [间接?] 如果或 $I_1(C)$ 或 $I_2(C)$ 等于 7,则置 C←M 并转到 A2。否则置 M←M + H, H←0。

A4. [降低深度] 如果 N > 0,则置 C←M, N←N - 1,并转到 A2。否则 M 是所求的答案。▎

这个算法将正确地处理任何情况,除了 $I_1 = 7$ 且 $1 \le I_2 \le 6$,以及在 ADDRESS 中地址的计算涉及 $I_1 = I_2 = 7$ 的一种状况外。效果就好像 I_2 为 0 一样。为了了解算法 A 的操作,考虑 a)部分的记号;在上述算法中,状态"L,7,1,2,5,2,7,7,7,7"通过 C 或 M = L, N = 4(末尾 7 的个数),以及 H = rI1 +

rI2 + rI5 + rI2 (后变址)来表示。在这道习题的 b)部分的解答中,计数器 N 将总是不为 0 就为 1。

6. (c)引起 OVERFLOW。(e) 引起 UNDERFLOW,而且如果这个程序继续,则它在最后的 I_2 上引起 OVERFLOW。

7. 否,因为 TOP[i]必须大于 OLDTOP[i]。

8. 对于一个栈,有用的信息出现在一端,而空信息出现在另一端:



其中 A = BASE[j], B = TOP[j], C = BASE[$j+1$]。对于一个队列或双端队列,有用的信息出现在两端,而空的信息出现在中间的某个地方:



或者有用的信息在当中而空信息在两端:



其中 A = BASE[j], B = REAR[j], C = FRONT[j], D = BASE[$j+1$]。这两种情况分别以在一个非空的队列中 $B \leqslant C$ 和 $B > C$ 的条件来区别之;或者,如果不知道队列是否已经溢出,则区分条件分别是 $B < C$ 和 $B \geqslant C$。因此应该以一种明显的方式来修改这些算法,以使空信息间隔加宽或缩小。(于是在溢出的情况下,即是当 $B = C$ 时,我们通过移动一部分而不移动另一部分,可做成 B 与 C 之间空的空间。)在步骤 G2 的 SUM 和 D[j]的计算中,应当把每个队列当作比它真正占有的单元要多占用一个单元(参见习题 1)。

9. 给定任何的序列说明 $a_1, a_2, \cdots, a_m$,对于每一个使得 $j < k$ 且 $a_j > a_k$ 的数偶(j,k),要求有一个移动操作。(这样的数偶称做一个"逆";见 5.1.1 小节。)因此,这样的数偶的个数是所需的移动次数。现在想像所有的 n^m 个说明都已写出,而且对于 $j < k$ 的$\binom{m}{2}$个数偶(j,k)中的每一对,计算有多少个有着 $a_j > a_k$ 的说明。显然,这就是$\binom{n}{2}$,即对于 a_j 和 a_k 之选择的个数,再乘以 n^{m-2},即填充剩下的位置的方式种数。因此在所有的说明当中,移动的总数为$\binom{m}{2}\binom{n}{2}n^{m-2}$。以 n^m 来除这个数即得到平均值,即等式(14)。

10. 如同在习题 9 中那样,我们求得预期的值是

$$\binom{m}{2}\sum_{1\leqslant j\leqslant k\leqslant n} p_j p_k = \frac{1}{2}\binom{m}{2}\left((p_1 + \cdots + p_n)^2 - (p_1^2 + \cdots + p_n^2)\right) =$$

$$\frac{1}{2}\binom{m}{2}\left(1 - (p_1^2 + \cdots + p_n^2)\right)$$

对于这一模型,它与这些表的相对次序是什么*绝对无差别*!(稍经思考即明白为什么;如果我们考虑给定的序列 $a_1, \cdots, a_m$ 的所有可能的排列,则我们发现,对所有这些排列求和的移动总数,仅仅依赖于不同元素 $a_j \neq a_k$ 的对偶的个数)。

11. 按前面来计算,我们求得预期的数是

$$E_{mnt}=\frac{1}{n^m}\binom{n}{2}\sum_{k=1}^{m}\sum_{r\geq t}(k-1)\binom{k-2}{r}(n-1)^{k-2-r}n^{m-k}$$

这里 r 是 $a_1,a_2,\cdots,a_{k-1}$之中等于 a_k 的元素的个数。此量也可简化为

$$E_{mnt}=\frac{1}{n^m}\binom{n}{2}\sum_{k>t}\binom{m}{k}(n-1)^{m-k}\left(\binom{k}{2}-\binom{t+1}{2}\right),\text{对于 } t\geqslant 0$$

是否还有更简单的方法来给出这个答案呢？显然没有了，因为对给定的 n 和 t，生成函数为

$$\sum_m E_{mnt}z^m=\frac{n-1}{2n}\frac{z}{(1-z)^3}\left(\frac{z}{n-(n-1)z}\right)^{t+1}(z+(1-z)n(t+1))$$

12. 如果 $m=2k$，则平均值是 2^{-2k}乘以

$$\binom{2k}{0}2k+\binom{2k}{1}(2k-1)+\cdots+\binom{2k}{k}k+\binom{2k}{k+1}(k+1)+\cdots+\binom{2k}{2k}2k$$

这后一个和式为

$$\binom{2k}{k}k+2\left(\binom{2k-1}{k}2k+\cdots+\binom{2k-1}{2k-1}2k\right)=\binom{2k}{k}k+4k\cdot\frac{1}{2}\cdot 2^{2k-1}$$

当 $m=2k+1$ 时，可使用类似的论证。答案为

$$\frac{m}{2}+\frac{m}{2^m}\binom{m-1}{\lfloor m/2\rfloor}$$

13. 姚期智证明了，当 $p<\frac{1}{2}$ 时，对于很大的 m，我们有 $\mathrm{E}\max(k_1,k_2)=\frac{1}{2}m+(2\pi(1-2p))^{-1/2}\sqrt{m}+O(m^{-1/2}(\log m)^2)$。[*SICOMP* **10** (1981), 398 ~ 403。]而且 P. Flajolet 推广了这个分析，并且特别地证明，当 $p=\frac{1}{2}$ 时，预期值渐近地为 αm，其中

$$\alpha=\frac{1}{2}+8\sum_{n\geq 1}\frac{\sin(n\pi/2)\cosh(n\pi/2)}{n^2\pi^2\sin n\pi}\approx 0.67531\ 44833$$

而且当 $p>\frac{1}{2}$，$m\to\infty$ 时，k_1 的最后的值倾向于一致地分布，所以 $\mathrm{E}\max(k_1,k_2)\approx\frac{3}{4}m$。[见*Lecture Notes in Comp. Sci.* **233** (1986), 325 ~ 340。]

14. 设 $k_j=m/n+\sqrt{m}x_j$。(这一想法是由 N. G. de Bruijn 提出的。)斯特林近似公式意味着，当 $k_1+\cdots+k_n=m$ 且当诸 x 一致地有界时，

$$n^{-m}\frac{m!}{k_1!\cdots k_n!}\max(k_1,\cdots,k_n)=$$

$$(\sqrt{2\pi})^{1-n}n^{n/2}\left(\frac{m}{n}+\sqrt{m}\ \max(x_1,\cdots,x_n)\right)\times$$

$$\exp\left(-\frac{n}{2}(x_1^2+\cdots+x_n^2)\right)(\sqrt{m})^{1-m}\left(1+O\left(\frac{1}{\sqrt{m}}\right)\right)$$

对于满足这个条件的所有非负的 $k_1,\cdots,k_n$，后边这个量之和近似于一个黎曼积分；我们可以导

出这个和的渐近性质,即 $a_n(m/n)+c_n\sqrt{m}+O(1)$,其中

$$a_n = (\sqrt{2\pi})^{1-n}n^{n/2}\int_{x_1+\cdots+x_n=0}\exp\left(-\frac{n}{2}(x_1^2+\cdots+x_n^2)\right)dx_2\cdots dx_n$$

$$c_n = (\sqrt{2\pi})^{1-n}n^{n/2}\int_{x_1+\cdots+x_n=0}\max(x_1,\cdots,x_n)\exp\left(-\frac{n}{2}(x_1^2+\cdots+x_n^2)\right)dx_2\cdots dx_n$$

因为有可能证明,对于任何 ϵ,对应的和进入 a_n 和 c_n 的 ϵ 之内。

我们知道 $a_n=1$,因为对应的和可明确地进行求值。出现于 c_n 的表达式中的积分等于 nI_1,其中

$$I_1 = \int_{\substack{x_1+\cdots+x_n=0\\ x_1\geqslant x_2,\cdots,x_n}}x_1\exp\left(-\frac{n}{2}(x_1^2+\cdots+x_n^2)\right)dx_2\cdots dx_n$$

我们可以作替换

$$x_1=\frac{1}{n}(y_2+\cdots+y_n),\ x_2=x_1-y_2,\ x_3=x_1-y_3,\ \cdots,\ x_n=x_1-y_n$$

然后我们求得 $I_1=I_2/n^2$,其中

$$I_2 = \int_{y_2,\cdots,y_n\geqslant 0}(y_2+\cdots+y_n)\exp\left(-\frac{Q}{2}\right)dy_2\cdots dy_n$$

而且 $Q=n(y_2^2+\cdots+y_n^2)-(y_2+\cdots+y_n)^2$ 现在由对称性,I_2 是$(n-1)$乘上这同一积分,但其中以 y_2 代替$(y_2+\cdots+y_n)$;因此 $I_2=(n-1)I_3$,其中

$$I_3 = \int_{y_2,\cdots,y_n\geqslant 0}(ny_2-(y_2+\cdots+y_n))\exp\left(-\frac{Q}{2}\right)dy_2\cdots dy_n =$$

$$\int_{y_3,\cdots,y_n\geqslant 0}\exp\left(-\frac{Q_0}{2}\right)dy_3\cdots dy_n$$

这里 Q_0 是以 0 代替 y_2 后的 Q。[当 $n=2$ 时,命 $I_3=1$。]现在设 $z_j=\sqrt{n}y_j-(y_3+\cdots+y_n)/(\sqrt{2}+\sqrt{n})$,$3\leqslant j\leqslant n$。于是 $Q_0=z_3^2+\cdots+z_n^2$,而且我们导出 $I_3=I_4/n^{(n-3)/2}\sqrt{2}$,其中

$$I_4 = \int_{y_3,\cdots,y_n\geqslant 0}\exp\left(-\frac{z_3^2+\cdots+z_n^2}{2}\right)dz_3\cdots dz_n =$$

$$\alpha_n\int\exp\left(-\frac{z_3^2+\cdots+z_n^2}{2}\right)dz_3\cdots dz_n = \alpha_n(\sqrt{2\pi})^{n-2}$$

其中 α_n 为$(n-2)$维空间中由向量$(n+\sqrt{2n},0,\cdots,0)-(1,1,\cdots,1),\cdots,(0,0,\cdots,n+\sqrt{2n})-(1,1,\cdots,1)$支起的"立体角",除以整个空间的总立体角。因此

$$c_n = \frac{(n-1)\sqrt{n}}{2\sqrt{\pi}}\alpha_n$$

我们有 $\alpha_2=1,\alpha_3=\frac{1}{2},\alpha_4=\pi^{-1}\arctan\sqrt{2}\approx .304$,而且

$$\alpha_5 = \frac{1}{8}+\frac{3}{4\pi}\arctan\frac{1}{\sqrt{8}} \approx .206$$

[c_3 的值是由 Robert M. Kozelka 求出的,*Annals of Math. Stat.* **27** (1956),507 ~ 512,但是,对于更高的 n 值,这一问题的解答,显然还从未在文献中出现过。]

16. 不能,除非队列满足应用于(4)和(5)的原始方法的限制。

17. 首先证明在所有时候都有 $\text{BASE}[j]_0 \leq \text{BASE}[j]_1$。然后注意到当栈 i 比以前更大,而它的新容量不比在 $s_1(\sigma)$中对于栈 i 原来分配的容量小时,会出现在 $s_0(\sigma)$中对于栈 i 的每个溢出,但并不引起 $s_1(\sigma)$中的溢出。

18. 假设一个插入的开销是 a,加上当需要重装时的 $bN+cn$,其中 N 是被占用的单元个数;设删除开销是 d。在使 N 个单元被占用和 $S=M-N$ 个单元为空的重装之后,想像每个插入直到下一个重装为止的开销是 $a+b+10c+10(b+c)nN/S=O(1+n\alpha/(1-\alpha))$,其中 $\alpha=N/M$。如果在该重装之前出现 p 个插入和 q 个删除,想像的开销是 $p(a+b+10c+10(b+c)nN/S)+qd$,而真正的开销是 $pa+bN'+cn+qd \leq pa+pb+bN+cn+qd$。后者小于想像的开销,因为 $p>.1S/n$;$M \geq n^2$ 的假定意味着 $cS/n+(b+c)N \geq bN+cn$。

19. 我们可以简单地把所有的下标都减 1;以下的解是比较漂亮的。开始时,T = F = R = 0。

把 Y 压入栈 X: 如果 T = M 则 OVERFLOW; X[T]←Y; T←T + 1。

从栈 X 弹出 Y: 如果 T = 0 则 UNDERFLOW; T←T - 1;Y←X[T]。

把 Y 插入队列 X: X[R]←Y; R←(R + 1)mod M; 如果 R = F,则 OVERFLOW。

从队列 X 删除 Y: 如果 F = R,则 UNDERFLOW; Y←X[F]; F←(F + 1)mod M。

如同以前一样,T 是栈中元素的个数,(R - F)mod M 是队列的元素个数。但栈顶元素现在是 X[T - 1],而不是 X[T]。

尽管对于计算机科学家来说从 0 计数几乎总是较好的,但是其他人大概不会改成以 0 开始的下标。甚至当 Edsger Dijkstra 在演奏钢琴时也在数着“1 - 2 - 3 - 4|1 - 2 - 3 - 4”!

2.2.3 小节

1. OVERFLOW 蕴涵在操作 P⇐AVAIL 中。

2.
```
INSERT  STJ   1F           存“NOP T”的单元
        STJ   9F           存出口单元
        LD1   AVAIL        rI1⇐AVAIL
        J1Z   OVERFLOW
        LD3   0,1(LINK)
        ST3   AVAIL
        STA   0,1(INFO)    INFO(rI1)←Y
1H      LD3   *(0:2)       rI3←LOC(T)
        LD2   0,3          rI2←T
        ST2   0,1(LINK)    LINK(rI1)←T
        ST1   0,3          T←rI1
9H      JMP   *            ▌
```

3.
```
DELETE  STJ   1F           存“NOP T”的单元
        STJ   9F           存出口单元
1H      LD2   *(0:2)       rI2←LOC(T)
        LD3   0,2          rI3←T
        J3Z   9F           是 T = Λ?
```

```
                LD1     0,3(LINK)     rI1←LINK(T)
                ST1     0,2           T←rI1
                LDA     0,3(INFO)     rA←INFO(rI1)
                LD2     AVAIL         AVAIL⇐rI3
                ST2     0,3(LINK)
                ST3     AVAIL
                ENT3    2             准备第二个出口
     9H         JMP     *,3           ▌
4. OVERFLOW     STJ     9F            存 rJ 的内容
                ST1     8F(0:2)       保留 rI1 的内容
                LD1     POOLMAX
                ST1     AVAIL         置 AVAIL 为新单元
                INC1    c
                ST1     POOLMAX       增加 POOLMAX
                CMP1    SEQMIN
                JG      TOOBAD        存储已超过?
                STZ     -c,1(LINK)    置 LINK(AVAIL)←Λ
     9H         ENT1    *             取 rJ 的设置
                DEC1    2             减 2
                ST1     *+2(0:2)      存出口单元
     8H         ENT1    *             恢复 rI1
                JMP     *             返回   ▌
```

5. 在前端的插入,实际上类似于基本插入操作(8),但却带有对于空队列的附加测试:P⇐AVAIL,INFO(P)←Y,LINK(P)←F;如果 F = Λ,则 R←P;F←P。

为了从后端删除,我们要找出哪个节点链接到 NODE(R),而且这必然是低效的,因为我们需要从 F 开始一路检索。例如,这可以按如下步骤进行:

a)如果 F = Λ,则 UNDERFLOW,否则置 P←LOC(F)。

b)如果 LINK(P)≠R,则置 P←LINK(P)并重复这一步骤直到 LINK(P) = R。

c)置 Y←INFO(R),AVAIL⇐R,R←P,LINK(P)←Λ。

6. 我们可以从(14)撤消操作 LINK(P)←Λ,如果我们从(17)删去命令"F←LINK(P)"和"如果 F = Λ 则置 R←LOC(F)"的话;后者代之以"如果 F = R,则 F←Λ 且 R←LOC(F),否则置 F←LINK(P)"。

这些改动的效果是,队列中的后端节点的 LINK 字段将包含不为程序所查询的虚假信息。类似于此的技巧,节省了执行时间,而且在实践中十分有用,尽管它违背了废料收集(见 2.3.5 小节)的基本假定之一,因而不能与这样的算法一起使用。

7.(要确保你的解答对于空表有效。)

I1. 置 P←FIRST,Q←Λ。

I2. 如果 P≠Λ,则置 R←Q,Q←P,P←LINK(Q),LINK(Q)←R,并重复这一步骤。

I3. 置 FIRST←Q。▌

实质上,我们是在把一些节点由一个栈弹出并把它们压入另一个栈。

```
8.              LD1     FIRST         1   I1. P≡rI1←FIRST
                ENT2    0             1   Q≡rI2←Λ
```

```
          J1Z     2F              1  I2.若表为空,则转移
1H        ENTA    0,2             n  R≡rA←Q
          ENT2    0,1             n  Q←P
          LD1     0,2(LINK)       n  P←LINK(Q)
          STA     0,2(LINK)       n  LINK(Q)←R
          J1NZ    1B              n  P≠Λ吗?
2H        ST2     FIRST           1  I3.FIRST←Q ▌
```

时间是$(7n+6)u$。可达到的更快速度为$(5n+常数)u$;参见习题 1.1-3。

9.(a) 是。(b) 是,如果考虑生物血统的话;否,如果考虑合法血统的话(像有一首歌所唱的"我是我自己的爷爷"那样,一个男人的女儿可能嫁给他的父亲)。(c) 否($-1<1$ 和 $1<-1$)。(d) 让我们这样希望,否则就会有循环论证。(e) $1<3$ 和 $3<1$。(f)这个命题是二义性的。如果我们采取这样的立场,即由 y 所调用的子程序依赖于调用 y 的程序,那我们就会得出这样的结论,即传递律不必成立。(例如,通用的输入输出子程序可以调用现有的每个输入输出设备的不同的处理程序,但通常在单个的程序中并不需要所有这些处理子程序。这乃是折磨许多自动程序设计系统的问题。)

10. 对于(i),有三种情况:$x=y$;$x\subset y$ 且 $y=z$;$x\subset y$ 且 $y\subset z$。对于(ii),有两种情况:$x=y$;$x\neq y$。这些情况的每一种均可平凡地处理之,像(iii)一样。

11."乘"下列数以得到全部 52 个解:13749(25 + 52)86 + (1379 + 1379 + 1937 + 9137)(4258 + 4528 + 2458 + 5428 + 2548 + 5248 + 2584 + 5284)6 + (1392 + 1932 + 1923 + 9123 + 9132 + 9213)7(458 + 548 + 584)6。

12. 例如:(a) 在具有 $k+1$ 个元素($0\leq k<n$)的所有集合之前,列出所有具有 k 个元素的集合(以任何次序)。(b) 通过表明哪一个元素在集合中的 0 和 1 的序列来表示一个子集。这通过二进制数系统给出了在所有子集与 0 到 2^n-1 之间的整数的对应关系。对应的次序就是一个拓扑序列。

13. Sha Jichang 和 D. Kleitman 在*Discrete Math.* **63** (1987), 271 ~ 278 中已经证明,这个数至多是 $\prod_{k=0}^{n}\binom{n}{k}^{\binom{n}{k}}$。这超过明显的下界 $\prod_{k=0}^{n}\binom{n}{k}!=2^{2^n(n+O(\log n))}$—个 $e^{2^n+O(n)}$ 的因子;他们猜测,这个下界是接近于真实的。

14. 如果 $a_1a_2\cdots a_n$ 和 $b_1b_2\cdots b_n$ 是两个可能的拓扑排序,令 j 是使得 $a_j\neq b_j$ 的极小值;于是对于某些 $k,m>j$, $a_k=b_j$ 和 $a_j=b_m$。现在 $b_j\not\preceq a_j$,因为 $k>j$,以及 $a_j\not\preceq b_j$,因为 $m>j$,因此(iv)不成立。反之,如果仅有一个拓扑排序 $a_1a_2\cdots a_n$,则对于 $1\leq j<n$,我们必定有 $a_j\preceq a_{j+1}$,因为否则 a_j 与 a_{j+1} 可加以交换。由此及传递性导出(iv)。

注:下面的另一种证明,也可对无穷集合进行。(a) 每个偏序均可嵌入到一个线性序中。因为如果我们有两个元素使得 $x_0\not\preceq y_0$ 和 $y_0\not\preceq x_0$,则我们通过规则"$x\preceq y$ 或 $x\preceq x_0$ 且 $y_0\preceq y$",可生成另一个偏序。这后一排序"包括"前一个,而且有 $x_0\preceq y_0$。现在在通常方式下应用Zorn引理或超穷归纳法来完成证明。(b) 显然,线性序不可能被嵌入到任何不同的线性序中。(c) 有着两个如同在(a)中那样的不可比较的元素 x_0 和 y_0 的偏序,可以扩充成两个线性序,其中分别有 $x_0\preceq y_0$ 和 $y_0\preceq x_0$。所以至少存在两个线性序。

15. 如果 S 是有限的,则我们可以列出所有在给定的偏序下为真的关系 $a<b$。通过逐次地撤消任何为其它所蕴涵的关系,一次一个,我们就得到一个非冗余的集合。问题是要证明,不论在什

么顺序下我们撤消这些冗余的关系,都恰有一个这样的集合。如果有两个非冗余的集合 U 和 V,其中"$a<b$"出现在 U 中而不在 V 中,则对于某个 $k\geq 1$,在 V 中有 $k+1$ 个关系 $a<c_1<\cdots<c_k<b$。但是有可能从 U 导出 $a<c_1$ 和 $c_1<b$,而无需使用关系 $a<b$(因为 $b\not\leq c_1$ 和 $c_1\not\leq a$),因此在 U 中关系 $a<b$ 是冗余的。

对于无穷的集合 S,当至多有一个非冗余的关系集合时,这个结果是不成立的。例如,如果 S 表示整数加上元素 ∞,而且我们定义:对于所有的 n,有 $n<n+1$ 和 $n<\infty$,则不存在表征这个偏序的非冗余的关系集合。

16. 设 $x_{p_1}x_{p_2}\cdots x_{p_n}$ 是 S 的一个拓扑排序;并把排列 $p_1\ p_2\cdots\ p_n$ 应用到行和列两者。

17. 如果在步骤 T4 中 k 从 1 增大到 n,则输出为 1932745860。如果在步骤 T4 中 k 从 n 减小到 1,如同它在程序 T 中那样,则输出为 9123745860。

18. 它们以排序后的顺序把诸项链接在一起:QLINK[0]为第一个,QLINK[QLINK[0]]是第二个,以此类推;QLINK[last] = 0。

19. 在某些情况下将失败;当这个队列在步骤 T5 中仅含一个元素时,修改的算法将置 F = 0(由此使队列变空),但其它条目在步骤 T6 中可放进这个队列。因此,建议的修改需在步骤 T6 中增加一项对 F = 0 的测试。

20. 确实,可以如下的方式使用一个栈(步骤 T7 消失):

T4. 置 T←0。对于 $1\leq k\leq n$,如果 COUNT[k]为 0,则做下列工作:置 SLINK[k]←T,T←k。(SLINK[k]≡QLINK[k]。)

T5. 输出 T 的值。如果 T = 0,则转到 T8;否则,置 N←N − 1,P←TOP[T],T←SLINK[T]。

T6. 和以前一样,但转到 T5 而不是 T7;而且当 COUNT[SUC(P)]减小为 0 时,置 SLINK[SUC(P)]←T和 T←SUC(P)。

21. 重复的关系仅使得这个算法稍微慢些,并且在存储池中花去更多的空间。关系"$j<j$"将被处理作类似一个循环(在对应的图中,一个从方框到它自身的箭头),它违背了偏序。

22. 为使程序"可靠",我们将(a)检验 $0<n<$ 某个适当的极大值;(b)对于条件 $0<j,k\leq n$,检验每个关系 $j<k$;(c)确保关系个数不溢出存储池区域。

23. 在步骤 T5 的末尾,增加"TOP[F]←Λ"。(于是,TOP[1],…,TOP[n]总是指向未被消去的所有关系。)在步骤 T8,如果 N > 0,则打印"LOOP DETECTED IN INPUT:",而且对于 $1\leq k\leq n$,置 QLINK[k]←0。现在增加以下步骤:

T9. 对于 $1\leq k\leq n$ 置 P←TOP[k],TOP[k]←0,并执行步骤 T10。(对于还未输出的每个 j,这将把 QLINK[j]置成对象 j 的前驱之一。)然后转到 T11。

T10. 如果 P≠Λ,置 QLINK[SUC(P)]←k,P←NEXT(P),并重复这一步骤。

T11. 找使得 QLINK[k]≠0 的 k。

T12. 置 TOP[k]←1 以及 k←QLINK[k]。现在如果 TOP[k] = 0,则重复此步。

T13. (我们已经找到了循环的开端。)打印 k 的值,置 TOP[k]←0,k←QLINK[k],而且如果 TOP[k] = 1,则重复此步。

T14. 打印 k 的值(这个循环的开始和末尾)并停机。(注:这个循环已被向后打印;如果希望以向前的次序打印这个循环,则应在步骤 T12 与 T13 之间使用类似于习题 7 中的算法。) ▌

24. 在正文的程序中插入三行:

```
08a  PRINTER  EQU  18
```

```
14a          ST6   NO
59a          STZ   X,1(TOP)          TOP[F]←Λ
```

以下列诸行来代替 74～75 行：

```
74           J6Z   DONE
75           OUT   LINE1(PRINTER)    打印循环标志
76           LD6   NO
77           STZ   X,6(QLINK)        QLINK[k]←0
78           DEC6  1
79           J6P   *-2               n≥k≥1
80           LD6   NO
81   T9      LD2   X,6(TOP)          P←TOP[k]
82           STZ   X,6(TOP)          TOP[k]←0
83           J2Z   T9A               P=Λ吗?
84   T10     LD1   0,2(SUC)          rI1←SUC(P)
85           ST6   X,1(QLINK)        QLINK[rI1]←k
86           LD2   0,2(NEXT)         P←NEXT(P)
87           J2P   T10               P≠Λ吗?
88   T9A     DEC6  1
89           J6P   T9                n≥k≥1
90   T11     INC6  1
91           LDA   X,6(QLINK)
92           JAZ   *-2               求使 QLINK[k]≠0 的 k
93   T12     ST6   X,6(TOP)          TOP[k]←k
94           LD6   X,6(QLINK)        k←QLINK[k]
95           LD1   X,6(TOP)
96           J1Z   T12               TOP[k]=0吗?
97   T13     ENTA  0,6
98           CHAR                    转换 k 成字母数字
99           JBUS  *(PRINTER)
100          STX   VALUE             打印
101          OUT   LINE2(PRINTER)
102          J1Z   DONE              当 TOP[k]=0 时停止
103          STZ   X,6(TOP)          TOP[k]←0
104          LD6   X,6(QLINK)        k←QLINK[k]
105          LD1   X,6(TOP)
106          JMP   T13
107  LINE1   ALF   LOOP              标题行
108          ALF   DETEC
109          ALF   TED I
110          ALF   N INP
111          ALF   UT:
```

```
LINE2   ALF                 逐次的各行
VALUE   EQU   LINE2 + 3
        ORIG  LINE2 + 24
DONE    HLT                 计算结束
X       END   TOPSORT   ▌
```

注:如果把关系 9<1 和 6<9 加到数据(18)中,这个程序将循环打印“9,6,8,5,9”。

26. 解答之一,是像下边这样分两个阶段处理:

阶段 1(在对每个需要使用的子程序标志 B = 1 或 2 的同时,使用 X 表格作为(顺序的)栈。)

A0. 对于 1≤J≤N,如果 B(X[J])≤0,置 B(X[J])←B(X[J]) + 2。

A1. 如果 N = 0,则转向阶段 2;否则置 P←X[N]而且 N 减 1。

A2. 如果|B(P)| = 1,则转向 A1,否则置 P←P + 1。

A3. 如果 B(SUB1(P)))≤0,则置 N←N + 1,B(SUB1(P))←B(SUB1(P)) + 2,X[N]←SUB1(P)。如果 SUB2(P)≠0 且 B(SUB2(P)≤0,则对 SUB2(P)执行一组类似的动作。转到 A2。▌

阶段 2(遍历表格并分配内存。)

B1. 置 P←FIRST。

B2. 若 P = Λ,置 N←N + 1,BASE(LOC(X[N]))←MLOC,SUB(LOC(X[N]))←0,并结束本算法。

B3. 如果 B(P) > 0,则置 N←N + 1,BASE(LOC(X[N]))←MLOC,SUB(LOC(X[N]))←P,MLOC←MLOC + SPACE(P)。

B4. 置 P←LINK(P)并返回 B2。▌

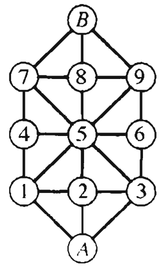
27. 关于下列代码的注释,留给读者来做。

```
B       EQU   0:1
SPACE   EQU   2:3
LINK    EQU   4:5
SUB1    EQU   2:3
SUB2    EQU   4:5
BASE    EQU   0:3
SUB     EQU   4:5
A0      LD2   N
        J2Z   B1
1H      LD3   X,2
        LDA   0,3(B)
        JAP   *+3
        INCA  2
        STA   0,3(B)
        DEC2  1
        J2P   1B
        LD1   N
A1      J1Z   B1
        LD2   X,1
        DEC1  1
A2      LDA   0,2(1:1)
        DECA  1
        JAZ   A1
        INC2  1
A3      LD3   0,2(SUB1)
        LDA   0,3(B)
        JAP   9F
        INC1  1
        INCA  2
        STA   0,3(B)
        ST3   X,1
9H      LD3   0,2(SUB2)
        J3Z   A2
        LDA   0,3(B)
        JAP   A2
        INC1  1
        INCA  2
        STA   0,3(B)
        ST3   X,1
        JMP   A2
B1      ENT2  FIRST
        LDA   MLOC
        JMP   1F
B3      LDX   0,2(B)
        JXNP  B4
        INC1  1
        ST2   X,1(SUB)
        ADD   0,2(SPACE)
1H      STA   X+1,1(BASE)
B4      LD2   0,2(LINK)
B3      J2NZ  B3
        STZ   X+1,1(SUB) ▌
```

28. 这里我们仅仅给出关于军棋的几点注释。设 A 是有三个士兵的游戏者,三个士兵的棋子开始于节点 A13 上;设 B 是另一个游戏者。在这项游戏中,A 必须"捕捉" B,而且如果 B 能够引起一个位置被重复两次,则我们就认为 B 是胜利者。然而为了避免把游戏的全部历史保持为这些位置的完整部分,我们应该把算法修改如下:首先把位置 157-4,789-B,359-6 标记为 B 移动"失败"的位置,并且应用所建议的算法。现在对于 A 来说,想法就是只移动到 B 失败的位置。但 A 也必须谨慎从事以免重复以前的动作。一个好的计算机游戏程序将使用随机数生成程序以便在有一种以上取胜动作存在时可从中进行选择,因此明显的技术将是使计算机扮演游戏者 A 一方,只在导致 B 处于失败位置的那些动作中随机选择。但是,有一些有趣的情况,它使这似乎千真万确的过程出错!例如,考虑对 A 移动的位置 258-7;这是获胜的位置。从位置 258-7 出发,游戏者 A 可以试探移动到 158-7(按照本算法,它是对于 B 来说失败的位置)。但然后 B 移到 158-B,而这迫使 A 移到 258-B。而后 B 就移回到 258-7;B 获胜,因为前边的位置已被重复!这个例子表明,在完成每个动作之后,必须重新调用这个算法,并且从以前已经被标记过"失败"(如果是 A 走)或者"胜利"(如果是 B 走)的每个位置开始。这一军棋可成为非常令人满意的计算机演示程序。

29. (a)如果 FIRST = Λ,则什么也不做;否则置 P←FIRST,而后反复地置 P←LINK(P)零次或多次,直到 LINK(P) = Λ 为止。最后置 LINK(P)←AVAIL 以及 AVAIL←FIRST(而且也许还有 FIRST←Λ)。(b)如果 F = Λ,则什么也不做;否则置 LINK(R)←AVAIL 以及 AVAIL←F(而且也许还有 F←Λ,R←LOC(F))。

30. 为了插入,置 P⇐AVAIL,INFO(P)←Y,LINK(P)←Λ,如果 F = Λ,则置 F←P 否则 LINK(R)←P 和 R←P。为了删除,以 F 代替 T 做(9)。(尽管对于空队列来说令 R 无定义是方便的,但这样不按规矩可能会像在习题 6 那样,引起废料收集算法的混乱。)

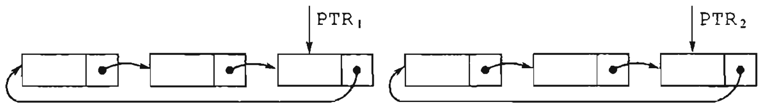


军棋的棋盘

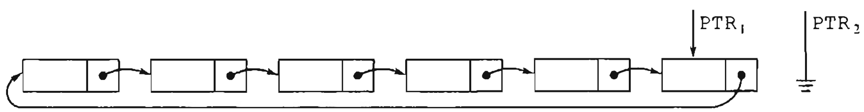
2.2.4 小节

1. 否,它并无帮助;似乎倒是妨碍(如果有的话)。(所述的约定不是与循环表原理特别相一致的,除非我们把 NODE(LOC(PTR))作为表头放入表中。)

2. 之前:



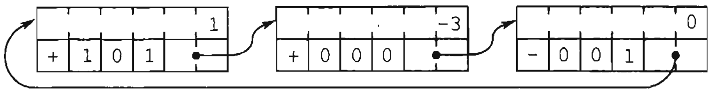
之后:

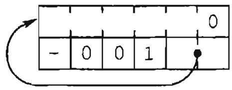


3. 如果 $PTR_1 = PTR_2$,则惟一的效果是 $PTR_2 \leftarrow \Lambda$。如果 $PTR_1 \neq PTR_2$,则链接的交换把表分成两个部分,就好像通过在两个点处切割而把圆分成两半似的。这个操作的第二部分使 PTR_1 指向一个循环表,这个循环表是由这样一些节点组成的,即如果在原来的表中,我们沿着从 PTR_1 到 PTR_2 的链接移动,这些节点将被遍历。

4. 令 HEAD 是表头的地址。为把 Y 压入栈中:置 P⇐AVAIL,INFO(P)←Y,LINK(P)←LINK(HEAD),LINK(HEAD)←P。为弹出栈到 Y 上:如果 LINK(HEAD) = HEAD 则 UNDERFLOW;否则置 P←LINK(HEAD),LINK(HEAD)←LINK(P),Y←INFO(P),AVAIL⇐P。

5.(同习题 2.2.3-7 作比较)置 Q←Λ,P←PTR,而后,当 P≠Λ 时,重复地置 R←Q,Q←P,P←LINK(Q),LINK(Q)←R。(然后 Q = PTR。)

6. (a) 

(b) 

7. 在对表的一次扫描中找出多项式中匹配的项的位置,因此就避免了重复的随机查找。而且,递增的次序将同"-1"的标志不相兼容。

8. 我们必须知道哪个节点指向当前感兴趣的节点,如果我们打算删去该节点或者在它的前边插入另一个节点的话。然而,也还有别的可能性:我们可以通过置 Q2←LINK(Q)而后置 NODE(Q)←NODE(Q2),AVAIL⇐Q2 来删除 NODE(Q);我们可以通过首先交换 NODE(Q2)↔NODE(Q),而后置 LINK(Q)←Q2,Q←Q2 而在 NODE(Q)的前边插入 NODE(Q2)。这些机智的技巧允许在不知道哪个节点链接到 NODE(Q)的情况下进行删除和插入;它们被用在 IPL 的早期版本中。但是它们却有这样一个缺点,即在多项式末尾的标志节点有时将移动,而其它链接变量可能指向这个节点。

9. 如同它应当的那样,对于 P = Q,算法 A 只不过使多项式(Q)加倍——除对于 ABC≥0 的某个项 COEF = 0 的异常情况外,那时它很快地失败。对于 P = M 的算法 M 也给出预料中的结果。如果 $M = t_1 + t_2 + \cdots + t_k$,对于 P = Q 的算法 M 置多项式(P)←多项式(P)乘以$(1+t_1)(1+t_2)\cdots(1+t_k)$(尽管这并非立即显然的)。当 M = Q 时,算法 M 令人惊讶地给出预料中的结果,多项式(Q)←多项式(Q) + 多项式(Q) × 多项式(P),除非当多项式(P)的常数项是 -1 时,这个计算崩溃。

10. 否。(惟一可能的不同将是在步骤 M2 中,并删除 A,B 或 C 可能个别地溢出的错误校验;这些错误校验未被描述,因为我们假定,它们并不是必要的。)换句话说,在本节中的算法可以当作是对于多项式 $f(x^{b^2}, x^b, x)$而不是对于 $f(x,y,z)$的操作。

11.(注释留给读者。)

```
COPY  STJ   9F                 ST6   1,3(LINK)
      ENT3  9F                 ENT3  0,6
      LDA   1,1                LD1   1,1(LINK)
1H    LD6   AVAIL              LDA   1,1
      J6Z   OVERFLOW           JANN  1B
      LDX   1,6(LINK)          LD2   8F(LINK)
      STX   AVAIL              ST2   1,3(LINK)
      STA   1,6           9H   JMP   *
      LDA   0,1           8H   CON   0  ▌
      STA   0,6
```

12. 设被复制的多项式有 p 项。程序 A 花费$(29p+13)u$,而且为使它成为公平的比较,我们应当添加建立零多项式的时间,比如说对于习题 14 的 $18u$,习题 11 的程序花费$(21p+31)u$。大约是$\frac{3}{4}$那么多。

```
13. ERASE  STJ   9F
           LDX   AVAIL
```

```
          LDA    1,1(LINK)
          STA    AVAIL
          STX    1,1(LINK)
    9H    JMP    *          ▌
```

14.
```
ZERO      STJ    9F
          LD1    AVAIL
          J1Z    OVERFLOW
          LDX    1,1(LINK)
          STX    AVAIL
          ENT2   0,1
          MOVE   1F(2)
          ST2    1,2(LINK)
    9H    JMP    *
    1H    CON    0
          CON    -1(ABC)    ▌
```

15.
```
MULT      STJ    9F         子程序入口
          LDA    5F         改变开关的设定
          STA    SW1
          LDA    6F
          STA    SW2
          STA    SW3
          JMP    *+2
    2H    JMP    ADD        M2. 乘循环
    1H    LD4    1,4(LINK)  M1. 下一个乘数。M←LINK(M)
          LDA    1,4
          JANN   2B         如果 ABC(M)≥0 则转到 M2
    8H    LDA    7F         恢复开关的设定
          STA    SW1
          LDA    8F
          STA    SW2
          STA    SW3
    9H    JMP    *          返回
    5H    JMP    *+1        SW1 的新设定
          LDA    0,1        COEF(P)
          MUL    0,4        rX←COEF(P)×COEF(M)
          LDA    1,1(ABC)   ABC(P)
          JAN    *+2
          ADD    1,4(ABC)   +ABC(M),如果 ABC(P)≥0
          SLA    2          移入 rA 的 0:3 字段
          STX    0F         保存 rX 以供在 SW2 和 SW3 中使用
          JMP    SW1+1
    6H    LDA    0F         SW2,SW3 的新设定
    7H    LDA    1,1        SW1 的通常设定
    8H    LDA    0,1        SW2,SW3 的通常设定
    0H    CON    0          临时存储   ▌
```

16. 设 r 是多项式(M)的项数。此子程序需要 $21pr+38r+29+27\sum m'+18\sum m''+27\sum p'+8\sum q'$时间单位,其中后边的几个求和指的是在程序 A 的 r 个活动期间对应的数量。多项式(Q)的项数对应程序 A 的每次激活上升 $p'-m'$。如果我们做一个并非不合理的假设,即 $m'=0$ 和 $p'=\alpha p$,其中 $0<\alpha<1$,则我们就得到分别等于 $0,(1-\alpha)pr,\alpha pr$ 以及 $rq_0'+\alpha p(r(r-1)/2)$的和,其中 q_0' 为 q' 在第一次迭代中之值。总计为 $4\alpha pr^2+40pr+4\alpha pr+8q_0'r+38r+29$。这一分析指出,乘式应该比被乘式的项数要少些,因为我们要经常地跳过多项式(Q)中不匹配的项。(关于更快的算法,见习题 5.2.3-29。)

17. 实际上没什么优点;使用各种类型的表的加法和乘法子程序,实际上都是相同的。ERASE 子程序(见习题 13)的效率,显然是惟一重要的差别。

18. 设节点 x_i 的链接字段包含 LOC(x_{i+1})$\oplus$LOC(x_{i-1}),其中"$\oplus$"表示"异或"。其它不可逆的操作,例如模指针字段大小的加法或减法,也可以使用。把两个相邻表头包括在循环表中,来帮助事情得以适当地开始,也是方便的。(这项巧妙技术的起源不得而知。)

2.2.5 小节

1. 在左边插入 Y:P⇐AVAIL;INFO(P)←Y;LLINK(P)←Λ;RLINK(P)←LEFT;如果 LEFT≠Λ,则 LLINK(LEFT)←P,否则 RIGHT←P;LEFT←P。置 Y 于最左边并删去:如果 LEFT=Λ,则 UNDERFLOW;P←LEFT;LEFT←RLINK(P);如果 LEFT=Λ 则 RIGHT←Λ,否则 LLINK(LEFT)←Λ;Y←INFO(P);AVAIL⇐P。

2. 考虑逐次(在同一端)进行若干次删除的情况。在每次删除之后,我们必须知道下次要删除什么。这意味着表中的链接之指向乃是背着表的该端。所以在两端删除就意味着链接必须在两个方向进行。另一方面,习题 2.2.4-18 解释了如何在单个链接字段中表示两个链接;在这种方法下,一般的双端队列操作就可能进行了。

3. 为了说明 CALLUP 对于 CALLDOWN 的无关性,注意,例如在表格 1 中的电梯在 0393~0444 的时间里不停在 2 层或 3 层上,尽管有人在等待着;这些人已经按了 CALLDOWN,但如果他们已经按了 CALLUP,则这电梯将停止。

为了说明 CALLCAR 与其它的无关性,注意在表 1 中,当门在 1378 的时间开始打开时,电梯已经判定为 GOINGUP。如果 CALLCAR[1]=CALLCAR[2]=CALLCAR[3]= CALLCAR[4]=0,则根据步骤 2,这时它在该点的状态应是 NEUTRAL,但是实际上,CALLCAR[2]和 CALLCAR[3]已由电梯中的第 7 人和第 9 人置为 1。(如果我们想像对于所有层号都增加 1 的同一情况,则当门打开时,STATE = NEUTRAL 或 STATE = GOINGUP 之事实,将影响电梯是否也许将继续下行还是无条件地上行)。

4. 如果有许多人离开同一层,则在所有这一时间里,STATE 可以是 NEUTRAL,而且当 E9 调用 DECISION 子程序时,在任何人抵达当前这一层之前,这可以置为一个新的状态。它确实是很少发生的(而且,它确实是作者在他进行的电梯实验期间所发现的最伤脑筋的现象)。

5. 由门在 1063 的时间开始打开之时起,直到第 7 人在时间 1183 抵达为止,状态应是 NEUTRAL,因为已经没有对第 0 层的呼叫而且没有人在电梯上。然后第 7 人将置 CALLCAR[2]←1 并且状态将相应地变成为 GOINGUP。

6. 对于步骤 U2 和 U4 中的条件"FLOOR=IN",加上条件"如果 OUT<IN 则 STATE≠GOINGUP;如果 OUT>IN 则 STATE≠GOINGDOWN"。在步骤 E4 中,从 QUEUE[FLOOR]接受乘客仅当他们的目标是和电梯的方向相一致时,除非 STATE = NEUTRAL(其时,我们接受所有的到来者)。

[斯坦福的数学系才有这样的电梯,但它的乘客实际上不大注意指示灯;人们只想尽快地进电梯,而不管它的方向。为什么电梯的设计者们不认识这一点,而相应地通过清除 CALLUP 和 CALL-

DOWN 来进行设计呢？整个过程将会更快些，因为那样的话电梯不会经常停下。]

7. 在 227 行，假定这个人是在 WAIT 表中。转移到 U4A 可确保这个假定成立。假定 GIVEUPTIME 为正，而且它确实是 100 以至更多。

8. 注释留给读者来做。

```
277   E8        DEC4    1
278             ENTA    61
279             JMP     HOLDC
280             LDA     CALL,4(3:5)
281             JAP     1F
282             ENT1    -2,4
283             J1Z     2F
284             LDA     CALL,4(1:1)
285             JAZ     E8
286   2H        LDA     CALL-1,4
287             ADD     CALL-2,4
288             ADD     CALL-3,4
289             ADD     CALL-4,4
290             JANZ    E8
291   1H        ENTA    23
292             JMP     E2A
```

9.

```
01   DECISION  STJ    9F                  存储出口位置
02             J5NZ   9F                  D1. 判定需要吗?
03             LDX    ELEV1+2(NEXTINST)
04             DECX   E1                  D2. 门应打开吗?
05             JXNZ   1F                  如果电梯不在 E1 处则转移
06             LDA    CALL+2
07             ENT3   E3                  如果有对 2 层的请求，则
08             JANZ   8F                    准备调度 E3
09   1H        ENT1   -4                  D3. 有任何请求吗?
10             LDA    CALL+4,1            查找一个非零调用变量
11             JANZ   2F
12   1H        INC1   1                   rI1≡j-4
13             J1NP   *-3
14             LDA    9F(0:2)             所有 CALL[j],j≠FLOOR 为零
15             DECA   E6B                 出口单元 = 行 250 吗?
16             JANZ   9F
17             ENT1   -2                  置 j←2
18   2H        ENT5   4,1                 D4. 置 STATE
19             DEC5   0,4                 STATE←j-FLOOR
20             J5NZ   *+2
21             JANZ   1B                  j = FLOOR 一般不被准许
```

```
          JXNZ  9F              D5. 电梯不动吗?
          J5Z   9F              如果不在 E1 处或如果 j = 2 则转移
          ENT3  E6              否则调度 E6
     8H   ENTA  20              等候 20 个时间单位
          ST6   8F(0:2)         保存 rI6
          ENT6  ELEV1
          ST3   2,6(NEXTINST)   置 NEXTINST 为 E3 或 E6
          JMP   HOLD            调度活动
     8H   ENT6  *               恢复 rI6
     9H   JMP   *               从子程序退出 ▎
```

11. 开始时令 LINK[k] = 0, $1 \leq k \leq n$, 且 HEAD = -1。在改变 V[k]的模拟步骤期间,如果 LINK[k] $\neq 0$ 则给出出错指示;否则置 LINK[k] ← HEAD, HEAD ← k 并且置 NEWV[k]成为 V[k]的新值。在每个模拟步骤之后,置 k ← HEAD, HEAD ← -1,并且重复下列操作零或多次直到 $k < 0$ 为止:置 V[k] ← NEWV[k], t ← LINK[k], LINK[k] ← 0, k ← t。

如果我们把 NEWV 和 LINK 字段包括在与一个变量字段 V 相关联的每个节点中,则显然这个方法可以容易地适合于分散的变量的情况。

12. WAIT 表有从左到右的删除,但插入是从右到左地排序进行的(因为查找很可能从该边更短些)。还有当我们不知道正被删去的节点的前驱或后继时,我们在若干个地方从所有的三个表删去节点,只有 ELEVATOR 表可以转换成单方向的表,而不损失多少效率。

注:在离散模拟程序中,使用非线性表作为 WAIT 表可能是更可取的,以便减少排序进入的时间。5.2.3 小节讨论了维护优先队列,即"最小进先出"表,以及诸如此类的表的一般问题。已经知道有好多种方法,其中当在表中有 n 个元素时,为进行插入或删除,只需要 $O(\log n)$ 个操作,尽管当 n 很小时,当然不需要使用这样的奇特方法。

2.2.6 小节

1. (这里下标是从 1 到 n,而不像在等式(6)中那样从 0 到 n。) LOC(A[J,K]) = LOC(A[0,0]) + $2n$J + 2K,其中 A[0,0]是实际上不存在的假定节点。如果置 J = K = 1,我们得到 LOC(A[1,1]) = LOC(A[0,0]) + $2n + 2$,所以答案可以以好几种方式来表示。LOC(A[0,0])可能为负这一事实已经导致在编译程序和装入程序中的许多错误。

2. LOC(A[$\mathrm{I}_1, \cdots, \mathrm{I}_k$]) = LOC(A[$0, \cdots, 0$]) + $\sum_{1 \leq r \leq k} a_r \mathrm{I}_r$ = LOC(A[$l_1, \cdots, l_k$]) + $\sum_{1 \leq r \leq k} a_r \mathrm{I}_r - \sum_{1 \leq r \leq k} a_r l_r$,其中 $a_r = c \prod_{r < s \leq k} (u_s - l_s + 1)$。

注:对于 C 这样的程序设计语言中出现的结构的推广,以及计算相关常数的简单算法,请见 P. Deuel, *CACM* **9** (1966), 344 ~ 347。

3. $1 \leq k \leq j \leq n$ 当且仅当 $0 \leq k-1 \leq j-1 \leq n-1$;所以在对于下界零推导出的所有公式中分别地以 $k-1, j-1, n-1$ 来代替 k, j, n。

4. LOC(A[J,K]) = LOC(A[0,0]) + nJ − J(J − 1)/2 + K。

5. 令 A0 = LOC(A[0,0])。至少有两个解,并假定 J 是在 rI1 中而 K 是在 rI2 中。(i) "LDA TA2,1:7",其中单元 TA2 + j 是"NOP j + 1 * j/2 + A0,2";(ii) "LDA C1,7:2",其中单元 C1 包含"NOP TA,1:7"而单元 TA + j 含"NOP j + 1 * j/2 + A0"。后者花费多一个的循环但不把表格约束在变址寄存器 2 中。

6. (a) $\mathrm{LOC}(A[I,J,K]) = \mathrm{LOC}(A[0,0,0]) + \binom{I+2}{3} + \binom{J+1}{2} + \binom{K}{1}$。

(b) $\mathrm{LOC}(B[I,J,K]) = \mathrm{LOC}(B[0,0,0]) +$

$$\binom{n+3}{3} - \binom{n+3-I}{3} + \binom{n+2-I}{2} - \binom{n+2-J}{2} + K - J$$

因此所述的形式在这个情况下也是可能的。

7. $\mathrm{LOC}(A[I_1,\cdots,I_k]) = \mathrm{LOC}(A[0,\cdots,0]) + \sum_{1\le r\le k}\binom{I_r+k-r}{1+k-r}$。见习题 1.2.6-56。

8.(由 P. Nash 提供的解)设对于 $0\le I\le n, 0\le J\le n+1, 0\le K\le n+2$ 可以定义 X[I,J,K]。我们可以令 A[I,J,K] = X[I,J,K];B[I,J,K] = X[J,I+1,K];C[I,J,K] = X[I,K,J+1];D[I,J,K] = X[J,K,I+2];E[I,J,K] = X[K,I+1,J+1];F[I,J,K] = X[K,J+1,I+2]。这个方案是最佳可能的,因为它把六个四面体数组的所有 $(n+1)(n+2)(n+3)$ 元素无重叠地组装在连续的位置中。证明:A 和 B 穷尽使 $k=\min(i,j,k)$ 的所有单元 X[i,j,k];C 和 D 穷尽使 $j=\min(i,j,k)\ne k$ 的所有单元;E 和 F 穷尽使 $i=\min(i,j,k)\ne j,k$ 的所有单元。

(如果任何人想要把 $m!$ 个广义四面体数组的元素组装在 $(n+1)(n+2)\cdots(n+m)$ 个连续的单元中,这个构造可推广到 m 维上。把排列 $a_1a_2\cdots a_m$ 同每个数组关联起来,并把它的元素存入 X[$I_{a_1}+B_1, I_{a_2}+B_2,\cdots, I_{a_m}+B_m$]中,其中 $B_1B_2\cdots B_m$ 是在习题 5.1.1-7 中定义的 $a_1a_2\cdots a_m$ 的反演表。)

9. **G1**. 把指针变量 P1,P2,P3,P4,P5,P6 分别置为表 FEMALE,A21,A22,A23,BLOND,BLUE 的第一个单元。以下假定,每个表的末端以链接 Λ 给出,而且 Λ 比任何其它链接都要小。如果 P6 = Λ,则停止(不幸,表是空的)。

G2. (下列诸动作可以做许多可能的排序;我们首先选择检查 EYES,然后依次检查 HAIR,AGE,SEX)。置 P5←HAIR(P5) 零次或多次直到 P5≤P6 为止。如果现在 P5 < P6,则转到步骤 G5。

G3. 置 P4←AGE(P4),如果有必要就重复,直到 P4≤P6 为止。类似地对 P3 和 P2 也做同样操作直到 P3≤P6 和 P2≤P6 为止。如果现在 P4,P3,P2 全都小于 P6,则转到 G5。

G4. 置 P1←SEX(P1)直到 P1≤P6 为止。如果 P1 = P6,则我们已经找到所希望的女孩中的一个,所以输出她的地址,P6(她的年龄可由 P2,P3 和 P4 的状态确定)。

G5. 置 P6←EYES(P6)。如果 P6 = Λ,则现在就停止;否则返回 G2。 ▎

这个算法是有趣的,但是对于这种查找来说这不是组织表的最好方法。

10. 请见 6.5 节。

11. 至多 200 + 200 + 3·4·200 = 2800 个字。

12. VAL(Q0) = c,VAL(P0) = b/a,VAL(P1) = d。

13. 在每个表的末尾有一个标志,这个标志在表被编序的某个字段上"比低",是方便的。可以使用直接的单向表,例如在 BASEROW[i]中仅保留 LEFT 链接和在 BASECOL[j]中仅保留 UP 链接。通过对算法 S 作如下修改:在 S2 中,在设置 J←COL(P0)之前测试是否 P0 = Λ;如果是,则置 P0←LOC(BASEROW[I0])并转到 S3。在 S3 中,测试是否 Q0 = Λ;如果是,则结束。步骤 S4 应类似于步骤 S2 修改。在 S5 中,测试是否 P1 = Λ;如果是,如同 COL(P1) < 0 那样操作。在 S6 中,测试是否 UP(PTR[J]) = Λ;如果是,就如同 ROW 字段是负的那样操作。

这些修改使得算法更为复杂，而且除表头中的 ROW 或 COL 字段外(在 MIX 的情况下，这根本不节省)，并不节省内存空间。

14. 人们可以首先把在主元行中有非 0 元素的那些列链接在一起，以使所有其它的列，随着我们把每行作为主元时可被跳过。主元列为 0 的那些行立即被跳过。

15. 命 rI1 ≡ PIVOT, J; rI2 ≡ P0; rI3 ≡ Q0; rI4 ≡ P; rI5 ≡ P1, X; LOC(BASEROW[i]) ≡ BROW + i; LOC(BASECOL[j]) ≡ BCOL + j; PTR[j] ≡ BCOL + j(1:3)。

```
ROW        EQU    0:3
UP         EQU    4:5
COL        EQU    0:3
LEFT       EQU    4:5
PTR        EQU    1:3
PIVOTSTEP  STJ    9F           子程序入口，rI1 = PIVOT
S1         LD2    0,1(ROW)     S1. 初始化
           ST2    I0           I0←ROW(PIVOT)
           LD3    1,1(COL)
           ST3    J0           J0←COL(PIVOT)
           LDA    =1.0=        浮点常数 1
           FDIV   2,1
           STA    ALPHA        ALPHA←1/VAL(PIVOT)
           LDA    =1.0=
           STA    2,1          VAL(PIVOT)←1
           ENT2   BROW,2       P0←LOC(BASEROW[I0])
           ENT3   BCOL,3       Q0←LOC(BASECOL[J0])
           JMP    S2
2H         ENTA   BCOL,1
           STA    BCOL,1(PTR)  PTR[J]←LOC(BASECOL[J])
           LDA    2,2
           FMUL   ALPHA
           STA    2,2          VAL(P0)←ALPHA × VAL(P0)
S2         LD2    1,2(LEFT)    S2. 处理主元行。P0←LEFT(P0)
           LD1    1,2(COL)     J←COL(P0)
           J1NN   2B           如果 J≥0，则处理 J
S3         LD3    0,3(UP)      S3. 找新行。Q0←UP(Q0)
           LD4    0,3(ROW)     rI4←ROW(Q0)
9H         J4N    *            如果 rI4 < 0，则出口
           CMP4   I0
           JE     S3           如果 rI4 = I0，则重复
           ST4    I(ROW)       I←rI4
           ENT4   BROW,4       P←LOC(BASEROW[I])
S4A        LD5    1,4(LEFT)    P1←LEFT(P)
S4         LD2    1,2(LEFT)    S4. 找新列。P0←LEFT(P0)
```

```
                 LD1     1,2(COL)        J←COL(P0)
                 CMP1    J0
                 JE      S4              如果 J = J0,则重复
                 ENTA    0,1
                 SLA     2               rA(0:3)←J
                 J1NN    S5
                 LDAN    2,3             如果 J < 0,则置 VAL(Q0)←
                 FMUL    ALPHA             - ALPHA × VAL(Q0)
                 STA     2,3
                 JMP     S3
        1H       ENT4    0,5             P←P1
                 LD5     1,4(LEFT)       P1←LEFT(P)
        S5       CMPA    1,5(COL)        S5. 找 I,J 元素
                 JL      1B              循环直到 COL(P1)≤J
                 JE      S7              如果 = ,则转 S7
        S6       LD5     BCOL,1(PTR)     S6. 插入 I,J 元素。rI5←PTR[J]
                 LDA     I               A(0:3)←I
        2H       ENT6    0,5             rI6←rI5
                 LD5     0,6(UP)         rI5←UP(rI6)
                 CMPA    0,5(ROW)
                 JL      2B              如果 ROW(rI5) > I 则转移
                 LD5     AVAIL           X⇐AVAIL
                 J5Z     OVERFLOW
                 LDA     0,5(UP)
                 STA     AVAIL
                 LDA     0,6(UP)
                 STA     0,5(UP)         UP(X)←UP(PTR[J])
                 LDA     1,4(LEFT)
                 STA     1,5(LEFT)       LEFT(X)←LEFT(P)
                 ST1     1,5(COL)        COL(X)←J
                 LDA     I(ROW)
                 STA     0,5(ROW)        ROW(X)←I
                 STZ     2,5             VAL(X)←0
                 ST5     1,4(LEFT)       LEFT(P)←X
                 ST5     0,6(UP)         UP(PTR[J])←X
        S7       LDAN    2,3             S7. 主元化。- VAL(Q0)
                 FMUL    2,2               × VAL(P0)
                 FADD    2,5               + VAL(P1)
                 JAZ     S8              如果失去有效位,则到 S8
                 STA     2,5             否则存入 VAL(P1)
                 ST5     BCOL,1(PTR)     PTR[J]←P1
```

```
77          ENT4   0,5             P←P1
78          JMP    S4A             P1←LEFT(P),到 S4
79   S8     LD6    BCOL,1(PTR)     S8. 删去 I,J 元素。rI6←PTR[J]
80          JMP    *+2
81          LD6    0,6(UP)         rI6←UP(rI6)
82          LDA    0,6(UP)
83          DECA   0,5             UP(rI6) = P1 吗?
84          JANZ   *-3             循环直到相等
85          LDA    0,5(UP)
86          STA    0,6(UP)         UP(rI6)←UP(P1)
87          LDA    1,5(LEFT)
88          STA    1,4(LEFT)       LEFT(P)←LEFT(P1)
89          LDA    AVAIL           AVAIL ⇐ P1
90          STA    0,5(UP)
91          ST5    AVAIL
92          JMP    S4A             P1←LEFT(P),到 S4  ▌
```

注:用第 4 章的约定,71 ~ 74 行实际上将被编码成

```
LDA 2,3; FMUL 2,2; FCMP 2,5; JE S8; STA TEMP; LDA 2,5; FSUB TEMP
```

而且在单元 0 中还有一个适当的参数 EPSILON。

17. 对于每行 i 和每个元素 $A[i,k]\neq 0$,加 $A[i,k]$ 乘以 B 的行 k 到 C 的行 i 上。在这样做时,只维持 C 的 COL 链接;在这以后,可很容易填入 ROW 链接。[A. Schoor, *Inf. Proc. Letters* **15** (1982), 87 ~ 89。]

18. 分别的列 3,1,2 中的三个主元步,分别产生

$$\begin{pmatrix} \frac{1}{3} & \frac{2}{3} & \frac{1}{3} \\ -\frac{2}{3} & -\frac{1}{3} & -\frac{2}{3} \\ -\frac{1}{3} & -\frac{2}{3} & -\frac{1}{3} \end{pmatrix}, \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{2}{3} & \frac{1}{2} & 1 \\ -\frac{1}{2} & -\frac{1}{2} & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ -2 & 1 & 1 \\ 1 & -2 & 0 \end{pmatrix}$$

经最后的置换之后,我们有答案

$$\begin{pmatrix} 1 & -2 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

20. $a_0 = \mathrm{LOC}(A[1,1]) - 3$, $a_1 = 1$ 或 2, $a_2 = 3 - a_1$。

21. 例如,$M \leftarrow \max(I,J)$, $\mathrm{LOC}(A[I,J]) = \mathrm{LOC}(A[1,1]) + M(M-1) + I - J$。(这些公式已由许多人独立地提出过。A. L. Rosenberg 和 H. R. Strong 提出了以下的 k 维的推广:$\mathrm{LOC}(A[I_1,\cdots,I_k]) = L_k$,其中 $L_1 = \mathrm{LOC}(A[1,\cdots,1]) + I_1 - 1$, $L_r = L_{r-1} + (M_r - 1)^r + (M_r - I_r) \times (M_r^{r-1} - (M_r - 1)^{r-1})$,而且 $M_r = \max(I_1,\cdots,I_r)$ [*IBM Tech. Disclosure Bull.* **14** (1972), 3026 ~ 3028]。关于这种类型的进一步结果,请见 *Current Trends in Programming Methodology* **4** (Prentilce-Hall, 1978), 263 ~ 311。

22. 根据组合数系统(习题 1.2.6-56),我们可以令

$$p(i_1,\cdots,i_k)=\binom{i_1}{1}+\binom{i_1+i_2+1}{2}+\cdots+\binom{i_1+i_2+\cdots+i_k+k-1}{k}$$

[*Det Kongelige Norske Videnskabers Selskabs Forhandlinger* **34** (1961), 8 ~ 9。]

23. 当矩阵从 J 列增长成 J + 1 列时，如果有 m 行，则令 c[J] = LOC(A[0, J]) = LOC(A[0,0]) + mJ 类似地，当我们建立行 I 时如果有 n 列，则令 r[I] = LOC(A[I,0]) = LOC(A[0, 0]) + nI。于是我们可以使用分配函数

$$\text{LOC(A[I,J])}=\begin{cases}1+c[\text{J}], & \text{如果 } c[\text{J}]\geqslant r[\text{I}]\\ \text{J}+r[\text{I}], & \text{其余情况}\end{cases}$$

不难证明，c[J]≥r[I]意味着 c[J]≥r[I] + J，而 c[J]≤r[I]意味着 c[J] + I≤r[I]；因此关系

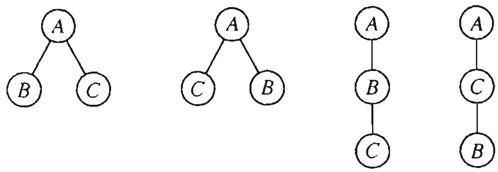
$$\text{LOC(A[I,J])}=\max(\text{I}+\text{LOC(A[0,J])},\text{J}+\text{LOC(A[I,0])})$$

也成立。我们不必限制分配于 mn 个连续的单元；惟一的限制是，当矩阵增长时，我们把 m 或 n 个连续的新单元分配到比以前已使用过的那些要大的新单元中。这个构造是由 E. J. Otoo 和 T. H. Merrett 给出的[*Computing* **31** (1983), 1 ~ 9]。他们也把它推广到 k 维上。

24. [Aho, Hopcroft 和 Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, 1974), 习题 2.12。]除了数组 A 外，还维护同样大小的验证数组 V 以及所使用单元的表 L。令 n 是 L 中的项数；开始时 $n=0$，而 L，A 和 V 的内容是任意的。每当对于你以前可能未使用过的 k 值要访问 A[k]时，首先检验是否 0≤V[k] < n 以及 L[V[k]] = k。如果不然，置 V[k]←n，L[n]←k，A[k]←0 以及 $n\leftarrow n+1$。否则你可确信，A[k]已经包含有合法的数据了。(对此方法稍加推广，有可能保存和最终恢复在计算期间改变了的 A 和 V 的所有条目的内容。)

2.3 节

1. 有三种方法来选择根。一旦已经选定了根，比如说 A，有三种方法把剩下的节点分成子树：$\{B\},\{C\}$；$\{C\},\{B\}$；$\{B,C\}$。在后一种情况下，有两种方法把$\{B,C\}$纳入一棵树，使用哪种方法依赖于哪个节点是它的根。因此，当 A 是根时，我们得到四棵树



而且全部有 12 棵树。在习题 2.3.4.4-23 中，对于任何节点数 n，都解决了这个问题。

2. 习题 1 答案中的前两棵树作为有向树而言是相同的，所以在这种情况下，我们仅得到 9 种不同的可能性。关于一般的解答，见 2.3.4.4 小节，其中证明了公式 n^{n-1}。

3. 部分 1：为证明至少有一个这样的序列。设树有 n 个节点。当 $n=1$ 时，结果是显然的，因为 X 必然是根。如果 $n>1$，则这定义蕴涵着有一个根 X_1 和子树 $T_1,T_2,\cdots,T_m$；或者 $X=X_1$，或者 X 是惟一的 T_j 之一成员。在后一种情况下，由归纳法，有一条通路 $X_2,\cdots,X$，其中 X_2 是 T_j 的根，而且由于 X_1 是 X_2 的父亲，我们仍有一条通路 $X_1,X_2,\cdots,X$。

部分 2：为证明至多有一个这样的序列。我们将通过归纳法证明，如果 X 不是树的根，则 X 有惟一的父亲(于是 X_k 确定 X_{k-1}，确定 X_{k-2}，等等)。如果树有一个节点，则没有什么可证的；否则 X 在惟一的 T_j 中。或者 X 是 T_j 的根，在这种情况下，由定义，X 有惟一的父亲；或者 X 不是 T_j 的根，

在这种情况下，由归纳法，X 在 T_j 中有惟一的父亲，而且在 T_j 之外没有节点能成为 X 的父亲。

4. 真的(很不幸)。

5. 4。

6. 设 $\text{parent}^{[0]}(X)$表示 X，并令 $\text{parent}^{[k+1]}(X)=\text{parent}(\text{parent}^{[k]}(X))$，使得 $\text{parent}^{[1]}(X)$是 X 的父亲，而 $\text{parent}^{[2]}(X)$是 X 的祖父；当 $k\geqslant 2$ 时，$\text{parent}^{[k]}(X)$是 X 的"$(\text{great})^{k-2}$祖父"。所要求的堂兄弟关系的条件是 $\text{parent}^{[m+1]}(X)=\text{parent}^{[m+n+1]}(Y)$但 $\text{parent}^{[m]}(X)\neq\text{parent}^{[m+n]}(Y)$。当 $n>0$ 时，这个关系对于 X 和 Y 不是对称的，尽管在日常的会话中通常把它当作是对称的。

7. 使用在习题 6 中定义的(非对称)条件，并且约定如果 j 或 k(或两者都在内)是 -1，则 $\text{parent}^{[j]}(X)\neq\text{parent}^{[k]}(Y)$。为了证明对于某个惟一的 m 和 n，这个关系总是成立的，考虑对于 X 和 Y 的杜威十进表示，即 $1.a_1.\cdots.a_p.b_1.\cdots.b_q$ 和 $1.a_1.\cdots.a_p.c_1.\cdots.c_r$，其中 $p\geqslant 0, q\geqslant 0, r\geqslant 0$ 而且(如果 $qr\neq 0$) $b_1\neq c_1$。任何节点对的杜威数都可以以这种形式写出，而且显然我们必须取 $m=q-1$ 和 $m+n=r-1$。

8. 没有二叉树是真正的树；这两个概念是完全独立的，尽管非空二叉树的图可能看起来像是树似的。

9. A 是根，因为我们习惯上把根放在顶上。

10. 嵌套集合的任何有限的汇集，如下对应于正文中所定义的那样的森林：设 $A_1,\cdots,A_n$ 是不含于其它当中的汇集之集合。对于固定的 j，包含于 A_j 中的所有集合的子汇集是嵌套的；因此我们可以假定这个子汇集对应于以 A_j 为根的(无序)树。

11. 在一个嵌套的汇集 $\mathscr{C}$ 中，令 $X\equiv Y$，如果有某个 $Z\in\mathscr{C}$ 使得 $X\cup Y\subseteq Z$ 的话。这一关系显然是反身的和对称的，而且事实上它是一个等价关系，因为 $W\equiv X$ 和 $X\equiv Y$ 意味着在 $\mathscr{C}$ 中有 Z_1, Z_2，同时 $W\subseteq Z_1, X\subseteq Z_1\cap Z_2, Y\subseteq Z_2$。由于 $Z_1\cap Z_2\neq\emptyset$，或者 $Z_1\subseteq Z_2$ 或者 $Z_2\subseteq Z_1$；因此 $W\cup Y\subseteq Z_1\cup Z_2\in\mathscr{C}$。现在如果 $\mathscr{C}$ 是一个嵌套的汇集，则通过规则"X 是 Y 的一个祖宗，而且 Y 是 X 的一个后裔，当且仅当 $X\supset Y$"定义一个对应于 $\mathscr{C}$ 的有向森林。$\mathscr{C}$ 的每个等价类对应于一个有向树，它是对于所有 X, Y 使得 $X\equiv Y$ 的有向森林。(如此我们就推广了森林和树的定义，这些对于有限的汇集是已经给出了的。)用这些术语，我们可以把 X 的*层次*定义作祖宗(X)的基数。类似地，X 的度数是在嵌套汇集后裔(X)中等价类的基数。我们说 X 是 Y 的*父亲*，以及 Y 是 X 的*儿子*，如果 X 是 Y 的祖宗，而且没有 Z 使得 $X\supset Z\supset Y$。(对 X 有可能有后裔但没有儿子，有祖宗但没有父亲。)为了得到有序树和森林，试以某种特别的方式将上述的等价类编序之，例如通过将关系 $\subseteq$ 嵌入到习题 2.2.3-14 中那样的线性序中。

a) 设 $S_{\alpha k}=\{x \mid x=$ 十进记号下的 $.d_1d_2d_3\cdots$，其中 $\alpha=$ 十进记号下的 $.e_1e_2e_3\cdots$，而且 $d_j=e_j$，如果 $j \bmod 2^k\neq 0\}$。汇集 $\mathscr{C}=\{S_{\alpha k}\mid k\geqslant 0, 0<\alpha<1\}$ 是嵌套的，并且给出一个树，具有无限多的层次，且对于每个节点具有不可数的度数。

b)，c) 在平面上而不是借助于实数，定义这样的集合是方便的，而且这是充分的，因为在平面与实数之间有一一对应关系。命 $S_{\alpha mn}=\{(\alpha,y)\mid m/2^n\leqslant y<(m+1)/2^n\}$，且令 $T_\alpha=\{(x,y)\mid x\leqslant\alpha\}$。汇集 $\mathscr{C}=\{S_{\alpha mn}\mid 0<\alpha<1, n\geqslant 0, 0\leqslant m<2^n\}\cup\{T_\alpha\mid 0<\alpha<1\}$ 容易看出是嵌套的。$S_{\alpha mn}$ 的儿子是 $S_{\alpha(2m)(n+1)}$ 和 $S_{\alpha(2m+1)(n+1)}$，而且 T_α 有儿子 $S_{\alpha 00}$ 加上子树 $\{S_{\beta mn}\mid\beta<\alpha\}\cup\{T_\beta\mid\beta<\alpha\}$。所以每个节点有度数 2，而且每个节点有不可数的形如 T_α 的祖宗。这个构造是由 R. Bigelow 给出的。

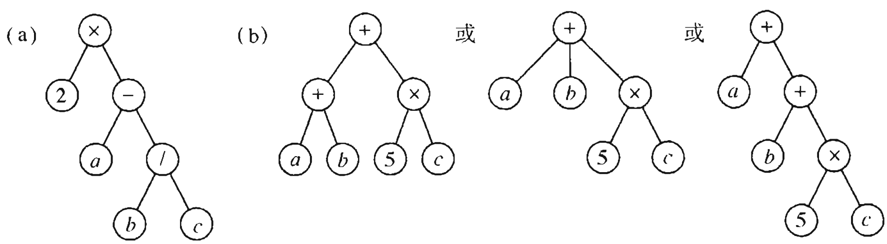
注：如果我们取实数的一个适当的良序，而且如果我们定义 $T_\alpha=\{(x,y)\mid x\succ\alpha\}$，则我们可以稍微地改进这个构造，得到一个嵌套的汇集，其中每个节点有不可数的层次，度数 2 以及 2 个儿子。

12. 我们对于偏序附加另外一个条件(类似于“嵌套集合”的条件)以保证它对应于一个森林:如果 $x \leq y$ 且 $x \leq z$,则或者 $y \leq z$ 或者 $z \leq y$。换句话说,大于任何给定元素的诸元素是线性有序的。为做成一个树,也断言对于所有的 x,存在一个最大的元素 r 使得对所有 x,$x \leq r$。当节点数有限时,这给出一个如同正文中所定义的那样的无序树,其证明像证明习题 10 中的嵌套集合那样进行。

13. a_1, $a_1.a_2$, $\cdots$, $a_1.a_2.\cdots.a_k$。

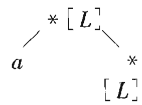
14. 由于 S 是非空的,它包含一个元素 $1.a_1.\cdots.a_k$,其中 k 尽可能小;如果 $k>0$,则我们也取 a_k 在 S 中尽可能小,而且我们立即看出,k 必须为 0。换句话说,S 必须含有元素 1。设 1 是根。所有其它的元素有 $k>0$,所以 S 的剩下的元素可以划分成集合 $S_j=\{1.j.a_2.\cdots.a_k\}$,对于某个 $m \geqslant 0$,$1 \leqslant j \leqslant m$。如果 $m \neq 0$,且 S_m 非空,则通过如上的推理,我们推出对于每个 S_j,$1.j$ 是在 S_j 中;因此每个 S_j 是非空的。于是容易看出,集合 $S_j' = \{1.a_2.\cdots.a_k \mid 1.j.a_2.\cdots.a_k$ 是在 S_j 中$\}$满足与 S 所满足的相同的条件。由归纳法,每个 S_j 也形成一棵树。

15. 设根为“1”而且设 α 的左子树的根和右子树的根分别为 $\alpha.0$ 和 $\alpha.1$,当这样的根存在时。例如在图 18(a)中,国王 Christian IX 出现在两个位置上,即 1.0.0.0.0 和 1.1.0.0.1.0。为了简便,我们可以去掉小数点,而仅仅写作 10000 和 110010。注:这个记号是由 Francis Galton 给出的;见*Natural Inheritance* (Macmillan,1889),249。对于“家系”,使用 F 和 M 来代替 0 和 1 并去掉开头的 1 是更好记的;例如,Christian IX 是 Charles 的*MFFMF*,即是 Charles 的母亲的父亲的父亲的母亲的父亲。0 和 1 的约定由于另一个原因,是有趣的:因为它向我们提供了重要的在二叉树的节点与以二进系统(即在计算机内的内存地址)表示的正整数之间的对应关系。



17. parent$(Z[1])=A$; parent$(Z[1,2])=C$; parent$(Z[1,2,2])=E$。

18. $L[5,1,1]=(2)$。$L[3,1]$是无意义的,因为 $L[3]$是一个空列表。

19. 

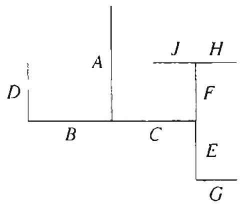
$L[2]=(L)$; $L[2,1,1]=a$。

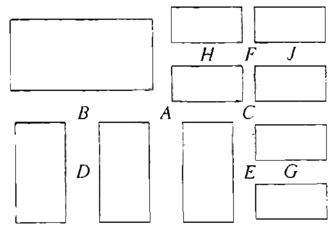
20.(直观上,0-2 树和二叉树之间的对应是通过删去 0-2 树的所有终端节点得到的;请见 2.3.4.5 小节中的重要构造。)设具有一个节点的 0-2 树对应于空的二叉树;并设多于一个节点,且因此由一个根 r 与 0-2 树 T_1 和 T_2 组成的一个 0-2 树对应于具有根 r,左子树 T_1' 和右子树 T_2' 的二叉树,其中 T_1 和 T_2 分别对应于 T_1' 和 T_2'。

21. $1+0\cdot n_1+1\cdot n_2+\cdots+(m-1)\cdot n_m$。证明:树中的节点数为 $n_0+n_1+n_2+\cdots+n_m$,而且这也等于 $1+$(树中的儿子数)$=1+0\cdot n_0+1\cdot n_1+2\cdot n_2+\cdots+m\cdot n_m$。

22. 基本思想是递归地进行,而且一个非空二叉树的表示被定义为它的根的表示加上它的左和右子树的大小折半和转动的表示。因此一个任意大的二叉树可以在一张纸上来表示,如果有充分大倍数的放大镜的话。

有可能有关于这个主题的许多变形。例如,一个想法是用从一张给定的横向纸张的中心到顶边的一条线来表示根,并且在左半页沿顺时针方向把左子树的表示转动 90°,在右半页按逆时针转动右子树 90°。于是每一节点由一条线表示。(当这个方法应用于在 k 层上有 2^k-1 个节点的完全二叉树时,就产生所谓的"H 树",它是在超大规模集成芯片上对于这种二叉树的最有效的布局;见 R. P. Brent 和 H. T. Kung, *Inf. Proc. letters* **11** (1980),46~48。)





另一个想法是通过某种框来表示空的二叉树,并且转动非空二叉树的子树表示,使得依赖于递归深度是偶数还是奇数,左子树交替地在对应的右子树的左边或下边。于是,这些框对应于扩充的二叉树(请见 2.3.4.5 小节)的外部节点。当外部节点携带信息而内部节点不带时,这个同在 6.5 节中讨论的 2-d 树以及四叉树紧密相关的表示,是特别适当的。

2.3.1 小节

1. INFO(T) = A, INFO(RLINK(T)) = C,等等;答案为 H。

2. 先根序:1245367;对称序:4251637;后根序:4526731。

3. 命题为真。注意,例如节点 4,5,6,7 在习题 2 中总以这种次序出现。通过对二叉树的大小用归纳法,即可证明这一结果。

4. 它是后根序的逆。(用归纳法容易证明该事实。)

5. 例如在习题 2 的树中,先根序为 1,10,100,101,11,110,111(用二进记号,在这种情况下等价于杜威系统)。如同在字典中的字那样,这些数字串已被排了序。

一般地说,节点将以先根序来列出,如果它们被从左到右地以字典序进行排序,而且有"空格" < 0 < 1。节点将以后根序列出,如果它们以 0 < 1 < "空格"的次序,按字典序进行排序。对于中根序,使用 0 < "空格" < 1。

(而且,如果我们想像空格在左边并且把杜威标号处理作通常的二进制数,我们就得到*层次序*;见 2.3.3-(8)。)

6. $p_1p_2\cdots p_n$ 可由一个栈得到这一事实,通过对 n 用归纳法,容易加以证明,否则事实上我们可以观察到,算法 T 精确地做着在其栈的动作中所要求做的工作。(如同在习题 2.2.1-3 中那样,诸 S 和诸 X 的对应序列,与在二重次序下的下标的诸 1 和诸 2 的序列是同样的,见习题 18。)

反之,如果 $p_1p_2\cdots p_n$ 可由栈得到,而且如果 $p_k=1$,则 $p_1\cdots p_{k-1}$ 是 $\{2,\cdots,k\}$ 的排列,而且 $p_{k+1}\cdots p_n$ 是 $\{k+1,\cdots,n\}$ 的排列;这些对应于左和右子树的排列,两者可都可由栈得到。现在可用归纳法来进行证明。

7. 由先根序,根就知道了;然后由中根序,我们知道左子树和右子树;而且事实上我们知道在后边的诸子树中节点的先根序与中根序。因此,该树容易被构造出来(而且事实上,构造这样一个简单的算法是十分有趣的:它以通常的方式把树链接在一起,开始于在 LLINK 中以先根序而在 RLINK 中以中根序链接在一起的节点)。类似地,后根序与中根序一起表征了这一结构。但先根

序与后根序则不然;有着以 AB 作为先根序和以 BA 作为后根序的两个二叉树。如果一个二叉树的所有非终端节点都有两个非空的分支,则它的结构即由先根序与后根序所表征。

8. (a)所有的 LLINK 为空的二叉树。(b) 具有 0 个或 1 个节点的二叉树。(c) 所有的 RLINK 为空的二叉树。

9. T1 一次,T2 $(2n+1)$次,T3 n 次,T4 $(n+1)$次,T5 n 次。这由归纳法或由基尔霍夫定律导出,或者通过验证程序 T 导出。

10. 所有 RLINK 为空的二叉树,将使得在撤消任何节点之前,把所有 n 个节点的地址都放进栈中。

11. 命 a_{nk}是具有 n 个节点的二叉树的个数,其中算法 T 中的栈绝不含有 k 个以上条目。如果 $g_k(z)=\sum_n a_{nk}z^n$,则求得 $g_1(z)=1/(1-z)$, $g_2(z)=1/(1-z/(1-z))=(1-z)/(1-2z)$, $\cdots$, $g_k(z)=1/(1-zg_{k-1}(z))=q_{k-1}(z)/q_k(z)$,其中 $q_{-1}(z)=q_0(z)=1$, $q_{k+1}(z)=q_k(z)-zq_{k-1}(z)$;因此 $g_k(z)=(f_1(z)^{k+1}-f_2(z)^{k+1})/(f_1(z)^{k+2}-f_2(z)^{k+2})$,其中 $f_j(z)=\frac{1}{2}(1\pm\sqrt{1-4z})$。现在可以证明,$a_{nk}=[u^n](1-u)(1+u)^{2n}(1-u^{k+1})/(1-u^{k+2})$;因此 $s_n=\sum_{k\geq 1}k(a_{nk}-a_{n(k-1)})$是$[u^{n+1}](1-u)^2(1+u)^{2n}\sum_{j\geq 1}u^j/(1-u^j)$,减去 a_{nn}。利用习题 5.2.2-52 中的技巧现在得到渐近序列

$$s_n/a_{nn}=\sqrt{\pi n}-\frac{3}{2}-\frac{13}{24}\sqrt{\frac{\pi}{n}}+\frac{1}{2n}+O(n^{-3/2})$$

[R. C. de Bruijn, D. E. Knuth 和 S. O. Rice, *Graph Theory and Computing*, R. C. Read 编辑(New York: Academic Press, 1972), 15 ~ 22。]

当二叉树如同 2.3.2 小节所述那样表示一个森林时,在这里所分析的量是该森林的高(即一个节点和一个根之间的最远距离加 1)。Flajolet 和 Odlyzko 已经得到对许多其它树的推广 [*J. Computer andSystem Sci.* **25** (1982), 171 ~ 213];高的渐近分布,包括靠近均值和远离均值在内,相继由 Flajolet,高志成,Odlyzko 及 Richmond 所分析 [*Combinatorics, Probability, and Computing* **2** (1993), 145 ~ 156]。

12. 在步骤 T2 与 T3 之间而不是在步骤 T5 访问 NODE(P)。为了证明,演示"在步骤 T2,以…原始值 A[1]…A[m]开始"的正确性,实际上和正文中一样。

13. (S. Araújo 的解) 设步骤 T1 到 T4 不变,但在步骤 T1 中把新变量 Q 初始化为 Λ;如果有的话,Q 将指向最后访问的节点。步骤 T5 变成两个步骤:

T5. [右分支完了?] 如果 RLINK(P) = Λ 或 RLINK(P) = Q,转向 T6;否则置 A⇐P, P←RLINK(P) 并返回 T2。

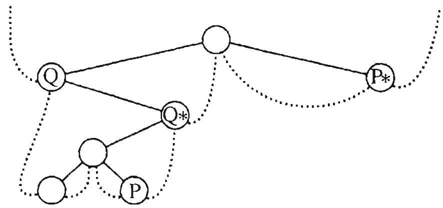
T6. [访问 P] 访问 NODE(P),置 Q←P,并返回 T4。

可应用类似的证明。(步骤 T4 和 T5 可以流水线化使得节点不用离开栈而立即重新插入。)

14. 由归纳法,总是恰有 $n+1$ 个 Λ 链接(当它为空时累计 T)。有 n 个非空的链接,累计 T,因此正文中关于空链居多的陈述是正确的。

15. 有一个指向一个节点的穿线的 LLINK 或 RLINK,当且仅当它分别具有一个非空的右子树或左子树。(见图 24。)

16. 如果 LTAG(Q) = 0,则 Q * 是 LLINK(Q);所以 Q * 是往左往下一步。否则,通过在树中往上重复(如果必要的话)进行,直到第一次有可能往右下方进行而不走回头路为止,才能得到Q *;典型的例子是在下列树中从 P 到 P * 和从 Q 到 Q * 的旅行:



17. 如果 LTAG(P) = 0,则置 Q←LLINK(P)并终止;否则置 Q←P,然后置 Q←RLINK(Q)零次或多次直到找到 RTAG(Q) = 0 为止。最后再次置 Q←RLINK(Q)。

18. 通过插入步骤 T2.5,“第一次访问 NODE(P)”,来修改算法 T;在步骤 T5 中,我们便第二次访问 NODE(P)。

给定穿线树,遍历是极为简单的:

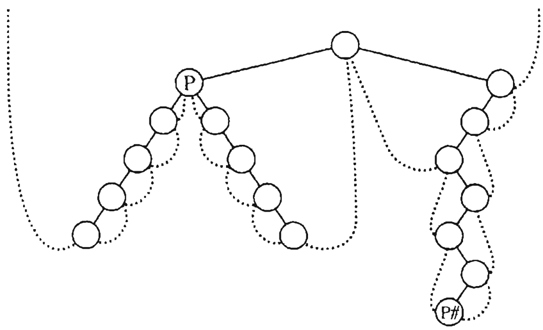
$(P,1)^{\Delta}$ = (LLINK(P),1),如果 LTAG(P) = 0,否则(P,2);

$(P,2)^{\Delta}$ = (RLINK(P),1),如果 RTAG(P) = 0,否则(RLINK(P),2)。

在每种情况下,我们在树中至多移动一步;所以实际上,双重次序和 d 与 e 之值已被嵌入程序中因而未明确提及。

删掉所有的第一次访问恰巧给予我们算法 T 和 S;删掉第二次访问给了我们习题 12 和 17 的解。

19. 基本思想是通过找 P 的父亲 Q 开始。然后如果 P≠LLINK(Q)我们有 P# = Q;否则通过重复地置 Q←Q $零次或多次直到 RTAG(Q) = 1 为止,我们可以找到 P#。(例如,请见以下所示树中的 P 和 P#。)



在一般的右穿线树中没有有效的求 P 的父亲的算法,因为所有左链接为空的退化的右穿线树实质上是链接错误的循环表。因此,如果我们不记住我们达到当前节点 P 的历史,则不能以和习题 13 的栈方法同样的效率,以后根序遍历一个右穿线树。

但是如果树是以两个方向穿线的,则我们能有效地找到 P 的父亲:

F1. 置 Q←P 和 R←P。

F2. 如果 LTAG(Q) = RTAG(R) = 0,则置 Q←LLINK(Q)和 R←RLINK(R)并重复这一步骤。否则如果 RTAG(R) = 1 则转到 F4。

F3. 置 Q←LLINK(Q)。如果 P = RLINK(Q)则结束。否则置 R←RLINK(R)零次或多次直到 RTAG(R) = 1 为止,然后置 Q←RLINK(R)并结束。

F4. 置 R←RLINK(R),而且如果 P = LLINK(R),则以 Q←R 结束。否则置 Q←LLINK(Q)零次或多次直到 LTAG(Q) = 1 为止,然后置 Q←LLINK(Q)并结束。 ▌

当 P 是树的随机节点时,算法 F 的平均运行时间是 $O(1)$。因为如果当 P 是右儿子时,我们仅统计步骤 Q←LLINK(Q),或者当 P 是左儿子时,我们仅统计步骤 R←RLINK(R),则每一个链接恰好被遍历一个节点 P。

20. 06 ~ 09 行代之以

```
T3   ENT4   0,6
     LD6    AVAIL
     J6Z    OVERFLOW
     LDX    0,6(LINK)
     STX    AVAIL
     ST5    0,6(INFO)
     ST4    0,6(LINK)
```

12 ~ 13 行代之以

```
     LD4    0,6(LINK)
     LD5    0,6(INFO)
     LDX    AVAIL
     STX    0,6(LINK)
     ST6    AVAIL
     ENT6   0,4
```

如果在行 06 处再加两行代码

```
T3   LD3    0,5(LINK)
     J3Z    T5              如果 LLINK(P) = Λ,则转 T5
```

并在行 10 和行 11 中作些适当的改变,则运行时间将从$(30n + a + 4)u$ 减少到$(27a + 6n - 22)u$。(如果我们置 $a = (n+1)/2$,则这同一设计将把程序 T 的运行时间减少到$(12a + 6n - 7)u$,这是一点改进。)

21. 下列的解是由 Joseph. M. Morris 提供的[*Inf. Proc. Letters* **9** (1979),197 ~ 200]。它也是以先根序进行遍历(见习题 18)。

U1.[初始化] 置 P←T 和 R←Λ。

U2.[完成了吗?] 如果 P = Λ,则算法结束。

U3.[向左找] 置 Q←LLINK(P)。如果 Q = Λ,则以先根序访问 NODE(P)并转到 U6。

U4.[查找穿线] 置 Q←RLINK(Q)零次或多次直到或者 Q = R 或者 RLINK(Q) = Λ 为止。

U5.[插入或删去穿线] 如果 Q≠R,则置 RLINK(Q)←P 并转到 U8。否则置 RLINK(Q)←Λ。(它已暂时被改成 P,但我们现在已遍历了 P 的左子树。)

U6.[中根序访问] 以中根序访问 NODE(P)。

U7.[向右或向上] 置 R←P,P←RLINK(P),并返回 U2。

U8.[先根序访问] 以先根序访问 NODE(P)。

U9.[转向左] 置 P←LLINK(P),并返回步骤 U3。 ▎

Morris 还提出了以后根序来进行遍历的稍复杂的方法。

J. M. Robson 找到了一个完全不同的方法[*Inf. Proc. Letters* **2** (1973),12 ~ 14]。我们说一个节点是“满”的,如果它的 LLINK 和 RLINK 都非空的话,说是“空”的,如果它的 LLINK 和 RLINK 都为空的话。Robson 找到维护指向满节点指针的栈的一个方法,利用在空节点中的链接字段,这些节点的右子树被访问。

还有另外一种避免辅助栈的方法,是由 G. Lindstrom 和 B. Dwyer 独立发现的,*Inf. Proc. Letter* **2** (1973),47 ~ 51,143 ~ 145。他们的算法以三重序遍历——它访问每个节点恰三次,以先根序,中根序和后根序各一次,但它不知道当前在做的是哪一个。

W1.[初始化] 置 P←T 和 Q←S,其中 S 是一个标志值——任何已知的不同于树中的任何链接的数(例如,-1)。

W2.[绕开空] 如果 P = Λ,置 P←Q 和 Q←Λ。

W3.［完成了吗?］如果 P = S,则结束此算法。(在结束时,我们将有 Q = T。)

W4.［访问］访问 NODE(P)。

W5.［旋转］置 R←LLINK(P),LLINK(P)←RLINK(P),RLINK(P)←Q,Q←P,P←R,并返回 W2。 ▎

正确性来自于如下事实,即如果我们在 W2 处开始,此时 P 指向二叉树 T 的根而 Q 指向 X,其中 X 不是树中的链接,则此算法将三次遍历树并且以 P = X 和 Q = T 达到步骤 W3。

如果 $\alpha(T) = x_1x_2\cdots x_{3n}$ 是在三重序下得到的节点序列,我们有 $\alpha(T) = T\alpha(LLINK(T))T\alpha(RLINK(T))T$。因此,如同 Lindstrom 所发现的那样,三个子序列 $x_1x_4\cdots x_{3n-2}$, $x_2x_5\cdots x_{3n-1}$, $x_3x_6\cdots x_{3n}$ 每个包括每个树的节点刚好一次。(由于 x_{j+1} 或者是 x_j 的父亲,或者是 x_j 的儿子,这些子序列是以这样一方式来访问这些节点的,即每个至多是由它的前驱引出的三个链接。7.2.1.6 小节会描述称做先后根序的一般遍历方案,该方案不但对二叉树而且对一般树也有这个性质。)

22. 这个程序使用程序 T 和 S 的约定,而且 Q 在 rI6 和/或 rI4 中。老式的 MIX 计算机不善于比较变址寄存器相等,因此省略了变量 R 并且把对于"Q = R"的测试改成"RLINK(Q) = P"。

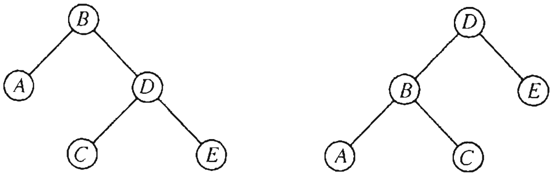
```
U1    LD5   HEAD(LLINK)   1               U1. 初始化。P←T
U2A   J5Z   DONE          1               如果 P = Λ 则停止
U3    LD6   0,5(LLINK)    n+a-1           U3. 向左找。Q←LLINK(P)
      J6Z   U6            n+a-1           如果 Q = Λ 则转到 U6
U4    CMP5  1,6(RLINK)    2n-2b           U4. 查找穿线
      JE    5F            2n-2b           如果 RLINK(Q) = P 则转移
      ENT4  0,6           2n-2b-a+1       rI4←Q
      LD6   1,6(RLINK)    2n-2b-a+1
      J6NZ  U4            2n-2b-a+1       以 Q←RLINK(Q)转 U4,如果它≠0
U5    ST5   1,4(RLINK)    a-1             U5a. 插入穿线。RLINK(Q)←P
U9    LD5   0,5(LLINK)    a-1             U9. 转向左。P←LLINK(P)
      JMP   U3            a-1             转到 U3
5H    STZ   1,6(RLINK)    a-1             U5b. 删去穿线。RLINK(Q)←Λ
U6    JMP   VISIT         n               U6. 中根序访问
U7    LD5   1,5(RLINK)    n               U7. 转向右或向上。P←RLINK(P)
U2    J5NZ  U3            n               U2. 完成了吗?如果 P≠Λ 则转到 U3
DONE  ...
```

总共的运行时间是 $21n + 6a - 3 - 14b$,其中 n 是节点数,a 是空的 RLINK 的个数(因此 $a - 1$ 是非空的 LLINK 的个数),而且 b 是在树的"右跨越"T,RLINK(T),RLINK(RLINK(T))上等的节点数。

23. 插入右边:RLINKT(Q)←RLINKT(P),RLINK(P)←Q,RTAG(P)←0,LLINK(Q)←Λ。插入左边,假定 LLINK(P) = Λ:置 LLINK(P)←Q,LLINK(Q)←Λ,RLINK(Q)←P,RTAG(Q)←1。在 P 与 LLINK(P)≠Λ 之间,插入左边:置 R←LLINK(P),LLINK(Q)←R,而后置 R←RLINK(R)零次或多次直到 RTAG(R) = 1 为止;最后,置 RLINK(R)←Q,LLINK(P)←Q,RLINK(Q)←P,RTAG(Q)←1。

(对最后的情况可以使用更有效的算法,如果知道一个节点 F,使得 P = LLINK(F)或 P = RLINK(F)的话;例如,假定是后者,则可置 INFO(P)↔INFO(Q),RLINK(F)←Q,LLINK(Q)←P,RLINK(Q)←RLINK(P),PLINK(P)←Q,RTAG(P)←1;这种运算只需固定的时间,但通常并不推荐使用,因为要在内存中交换节点。)

24. 否:



25. 我们首先对 T 中节点数用归纳法证明(b),并类似地证明(c)。现在(a)分成为好些情况;写 $T \le_1 T'$ 如果 i)成立,$T \le_2 T'$ 如果 ii)成立,等等。则 $T \le_1 T'$ 和 $T' \le T''$ 意味着 $T \le_1 T''$;$T \le_2 T'$ 和 $T' \le T''$ 意味着 $T \le_2 T''$;而剩下的两种情况通过对于 T 中的节点数用归纳法证明(a)来进行处理。

26. 如果 T 的双重序是$(u_1, d_1), (u_2, d_2), \cdots, (u_{2n}, d_{2n})$,其中之诸 u 是节点,诸 d 是 1 或 2,则构造该树的"迹"$(v_1, s_1), (v_2, s_2), \cdots, (v_{2n}, s_{2n})$,这里 $v_j = \text{info}(u_j)$,且 $s_j = l(u_j)$或 $r(u_j)$,依据 $d_j = 1$ 或 2。现在 $T \le T'$,当且仅当 T 的(按这里所定义的)迹按字典序居前于或等于 T' 的迹。形式上,这意味着对于 $1 \le j \le 2n$,或者 $n \le n'$ 且$(v_j, s_j) = (v'_j, s'_j)$,或者有一个 k,使得对于 $1 \le j < k$,$(v_j, s_j) = (v'_j, s'_j)$,而且或者 $v_k < v_k'$ 或者 $v_k = v'_k$ 且 $s_k = s'_k$。

27. **R1**. [初始化] 置 P←HEAD,P'←HEAD';这些分别是给定右穿线二叉树的表头。转到 R3。

R2. [校验 INFO] 如果 INFO(P) < INFO(P'),则终止($T < T'$);如果 INFO(P) > INFO(P'),则终止($T > T'$)。

R3. [转向左边] 如果 LLINK(P) = Λ = LLINK(P'),则转到 R4;如果 LLINK(P) = Λ ≠ LLINK(P'),则终止($T < T'$);如果 LLINK(P) ≠ Λ = LLINK(P'),则终止($T > T'$);否则置 P←LLINK(P),P'←LLINK(P')并转到 R2。

R4. [树结束?] 如果 P = HEAD(或者,等价地,如果 P' = HEAD'),则终止(T 等价于 T')。

R5. [转向右边] 如果 RTAG(P) = 1 = RTAG(P'),则置 P←RLINK(P),P'←RLINK(P'),并转到 R4。如果 RTAG(P) = 1 ≠ RTAG(P),则终止($T < T'$)。如果 RTAG(P) ≠ 1 = RTAG(P'),则终止($T > T'$)。否则,置 P←RLINK(P),P'←RLINK(P'),并转到 R2。▌

为证明这个算法的正确性(且因此了解它是怎样工作的),人们可以对树 T_0 的大小用归纳法来证明下列命题成立:"*从步骤 R2,且以 P 和 P'指向两个非空右穿线二叉树 T_0 和 T'_0 开始,如果 T_0 与 T'_0 不等价则算法将终止,并指出是 $T_0 < T'_0$ 还是 $T_0 > T'_0$;如果 T_0 与 T'_0 等价,则这个算法将到达步骤 R4,同时 P 和 P'于是分别指向在对称序下的后继节点 T_0 和 T'_0。*"

28. 等价而且相似。

29. 通过对 T 的大小用归纳法,证明下列命题是正确的:"*在步骤 C2 以 P 指向非空的二叉树 T 之根,且以 Q 指向有着非空的左和右子树的节点开始,在置 INFO(Q)←INFO(P)和对 NODE(Q)附加上 NODE(P)的左和右子树的副本之后,这个过程将最终地到达步骤 C6,而且 P 和 Q 分别指向树 T 和 NODE(Q)的先根序的后继节点。*"

30. 假定(2)中的指针 T 是(10)中的 LLINK(HEAD)。

L1. [初始化] 置 Q←HEAD,RLINK(Q)←Q。

L2. [推进] 置 P←Q\$(见以下)。

L3. [穿线] 如果 RLINK(Q) = Λ,则置 RLINK(Q)←P,RTAG(Q)←1;否则置 RTAG(Q)←0。如果 LLINK(P) = Λ,则置 LLINK(P)←Q,LTAG(P)←1;否则置 LTAG(P)←0。

L4. [完成了?] 如果 P ≠ HEAD,则置 Q←P 并返回到 L2。▌

这个算法的步骤 L2 意味着,中根序遍历共行程序的激活类似于算法 T,附加额外的条件:算法 T 在其完全遍历树之后访问 HEAD。这个记号在描述树的算法中是方便的简化,因为我们不需要一次又一次地重复算法 T 的栈机制。当然算法 S 在步骤 L2 期间不能使用,因为这个树尚未被穿线。但是,习题 21 的算法在步骤 L2 中能被采用,而且这为我们提供了一种不需要任何辅助栈来穿线树的非常巧妙的方法。

31. **X1**. 置 P←HEAD。

X2. 置 Q←P $(使用,比如说,对于右穿线树修改了的算法 S)。

X3. 如果 P≠HEAD,则置 AVAIL⇐P。

X4. 如果 Q≠HEAD,则置 P←Q 而转回到 X2。

X5. 置 LLINK(HEAD)←Λ。 ▌

减少内循环长度的其它解决方法显然是可能的,尽管基本步骤的次序是较关键性的。所述的过程有效,因为直到算法 S 已经考察过了它的 LLINK 和 RLINK 之后,我们才把一个节点返回到可用存储中。如同在正文中所观察过的那样,在完整的树遍历期间这些链接的每个恰好被使用一次。

32. RLINK(Q)←RLINK(P), SUC(Q)←SUC(P), SUC(P)←RLINK(P)←Q, PRED(Q)←P, PRED(SUC(Q))←Q.

33. 把 NODE(Q)刚好插入到 NODE(P)的左边和下边十分简单:置 LLINKT(Q)←LLINKT(P), LLINK(P)←Q, LTAG(P)←0, RLINK(Q)←Λ。对于右边的插入是颇为困难的,因为实际上要求寻找 *Q,这可以与寻找 Q# 的困难相比拟(见习题 19);也许可以使用习题 23 中讨论的移动节点的技术。所以对于这种类型的穿线一般的插入更为困难。但是算法 C 所要求的插入并不像一般的插入那样困难,而且事实上对于这种类型的穿线,复制过程倒更快些:

C1. 置 P←HEAD, Q←U,转到 C4。(始终地使用正文中算法 C 的假定和原理。)

C2. 如果 RLINK(P)≠Λ,则置 R⇐AVAIL, LLINK(R)←LLINK(Q), LTAG(R)←1, RLINK(R)←Λ, RLINK(Q)←LLINK(Q)←R。

C3. 置 INFO(Q)←INFO(P)。

C4. 如果 LTAG(P)=0 则置 R⇐AVAIL, LLINK(R)←LLINK(Q), LTAG(R)←1, RLINK(R)←Λ, LLINK(Q)←R, LTAG(Q)←0。

C5. 置 P←LLINK(P), Q←LLINK(Q)。

C6. 如果 P≠HEAD,则转到 C2。 ▌

这个算法现在看起来太简单了,以致不一定正确!

由于步骤 C5 中计算 P*, Q* 的额外时间,对于穿线或右穿线的二叉树,算法 C 花费的时间稍微长些。

在步骤 C2 和 C4 中适当地置 RLINK(R)和 RLINK(Q),同上述的复制方法相结合,可能以通常的方式对 RLINK 进行穿线或者置 #P 于 RLINK(P)中。

34. **A1**. 置 Q←P,然后重复地置 Q←RLINK(Q)零次或多次,直到 RTAG(Q)=1 为止。

A2. 置 R←RLINK(Q)。如果 LLINK(R)=P,则置 LLINK(R)←Λ;否则置 R←LLINK(R),然后重复地置 R←RLINK(R)零次或多次直到 RLINK(R)=P 为止;于是最后置 RLINKT(R)←RLINKT(Q)。(这个步骤已从原来的树撤消了 NODE(P)和它的子树。)

A3. 置 RLINK(Q)←HEAD, LLINK(HEAD)←P。 ▌

(想出和/或了解这个算法的关键,是构造好的“之前和之后”图式。)

36. 否;参看习题 1.2.1-15(e)的答案。

37. 如果在表示(2)中 LLINK(P) = RLINK(P) = Λ,则令 LINK(P) = Λ;否则令 LINK(P) = Q,其中 NODE(Q)对应于 NODE(LLINK(P)),而且 NODE(Q + 1)对应于 NODE(RLINK(P))。条件 LLINK(P)或 RLINK(P) = Λ分别通过 NODE(Q)或 NODE(Q + 1)中的一个标志来表示。这个表示使用 n 和 $2n+1$ 之间的内存单元;在所述假定下,(2)将需要 18 个内存字,同现在的方案的 11 个相对照。在每种表示下,插入和删除操作有近乎相等的效率。但这个表示在与其它结构相结合时,不那么适用。

2.3.2 小节

1. 如果 B 为空,则 $F(B)$是一个空的森林。否则,$F(B)$由一棵树 T 加上森林F(右子树(B))组成,其中 root(T) = root(B)而且子树(T) = F(左子树(B))。

2. 在二进记号下的 0 的个数,是在十进记号下小数点的个数;对应的精确公式是

$$a_1.a_2.\cdots.a_k \leftrightarrow 1^{a_1}01^{a_2-1}0\ldots01^{a_k-1}$$

其中 1^a 表示在一行中有 a 个 1。

3. 把节点的杜威十进记号按字典序(如同在字典中那样,从左到右)进行排序,对于先根序,把较短的序列 $a_1.\cdots.a_k$ 放在它的扩展 $a_1.\cdots.a_k.\cdots.a_r$ 的前边,而对于后根序,则放在它的扩展的后边。(于是,如果我们正在对字,而不是对数的序列进行排序,则对于先根序,将把字 *cat*, *cataract* 以通常的字典序放置,但对于后根序,则以相反于初始子字的次序来放置(*cataract*, *cat*)。对树的大小用归纳法,容易证明这些规则。

4. 通过对节点个数用归纳法,为真。

5. (a)中根序。(b)后根序。阐述这些遍历算法等价性的严格的归纳证明,是颇令人感兴趣的。

6. 我们有先根序(T) = 先根序(T'),后根序(T) = 中根序(T'),即使 T 有着仅有一个儿子的节点亦然。剩下的两个顺序不处于任何简单关系之中;例如,在一种情况下 T 的根出现于末尾,而在另一种情况下则差不多出现于中间。

7. (a)是;(b)否;(c)否;(d)是。注意森林的颠倒的先根序,等于左右颠倒的森林的后根序(在镜面反射的意义下)。

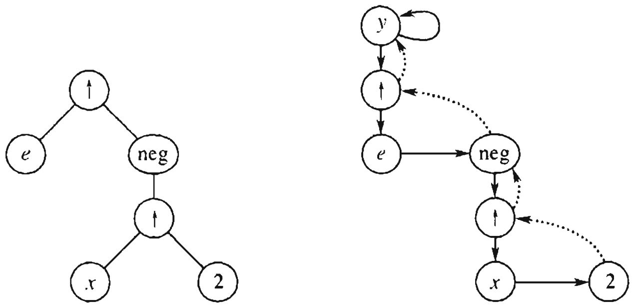
8. $T \preceq T'$意味着 info(root(T)) ≺ info(root(T')),或者这些 info 相等且下列条件成立:假设 root(T)的子树是 $T_1, \cdots, T_n$,而 root(T')的子树是 $T'_1, \cdots, T'_{n'}$,并且设 $k \geqslant 0$ 尽可能地大,使得对于 $1 \leqslant j \leqslant k$,$T_j$ 等价于 T'_j。于是或者 $k = n$ 或者 $k < n$ 且 $T_{k+1} \preceq T'_{k+1}$。

9. 在非空的森林中,非终端节点的个数,比其为 Λ 的右链接的个数小 1,因为空的右链接对应于每个非终端节点的最右儿子。而且也对应于森林中最右树的根(此事实给出了习题 2.3.1-14 的另一个证明,因为显然空的左链接的个数等于终节点的个数)。

10. 这些森林是相似的,当且仅当 $n = n'$ 且 $d(u_j) = d(u'_j)$, $1 \leqslant j \leqslant n$;它们是等价的,当且仅当再加上条件 info($u_j$) = info($u'_j$), $1 \leqslant j \leqslant n$。通过推广引理 2.3.1P,这个证明类似于以前的证明;令 $f(u) = d(u) - 1$。

11. 见下页图。

12. 如果 INFO(Q1) ≠ 0:置 R←COPY(P1);然后如果 TYPE(P2) = 0 且 INFO(P2) ≠ 2,则置 P←TREE(“↑”,TREE(INFO(P2) - 1));如果 TYPE(P2) ≠ 0,则置 R←TREE(“↑”,R,TREE(“-”,



COPY(P2),TREE(1)));然后置 Q1←MULT(Q1,MULT(COPY(P2),R))。

如果 INFO(Q) ≠ 0: 置 Q←TREE("×",MULT(TREE("ln",COPY(P1)),Q),TREE("↑",COPY(P1),COPY(P2)))。

最后转到 DIFF[4]。

13. 下列程序,以 rI1≡P,rI2≡Q,rI3≡R 实现算法 2.3.1C,并且在初始条件和终止条件中作适当的改变:

```
        ST3    6F(0:2)        保留 rI3,rI2 的内容
        ST2    7F(0:2)        C1. 初始化
        ENT2   8F             通过 RLINK(U) = Λ 建立
        JMP    1F               NODE(U)开始
8H      CON    0              进行初始化的常数 0
4H      LD1    0,1(LLINK)     置 P←LLINK(P) = P*
1H      LD3    AVAIL          R⇐AVAIL
        J3Z    OVERFLOW
        LDA    0,3(LLINK)
        STA    AVAIL
        ST3    0,2(LLINK)     LLINK(Q)←R
        ENNA   0,2
        STA    0,3(RLINK)     RLINK(R)←Q,RTAG(R)←1
        INCA   8B             rA←LOC(初始节点) - Q
        ENT2   0,3            置 Q←R = Q*
        JAZ    C3             转到 C3,头一次
C2      LDA    0,1            C2. 右边有什么?
        JAN    C3             如果 RTAG(P) = 1 则转移
        LD3    AVAIL          R⇐AVAIL
        J3Z    OVERFLOW
        LDA    0,3(LLINK)
        STA    AVAIL
        LDA    0,2(RLINKT)
        STA    0,3(RLINKT)    置 RLINKT(R)←RLINKT(Q)
        ST3    0,2(RLINKT)    RLINKT(Q)←R,RTAG(Q)←0
```

```
089  C3   LDA    1,1            C3. 复制 INFO
090       STA    1,2            复制的 INFO 字段
091       LDA    0,1(TYPE)
092       STA    0,2(TYPE)      复制的 TYPE 字段
093  C4   LDA    0,1(LLINK)     C4. 左边有什么?
094       JANZ   4B             如果 LLINK(P)≠Λ,则转移
095       STZ    0,2(LLINK)     LLINK(Q)←Λ
096  C5   LD2N   0,2(RLINKT)    C5. 推进。Q←-RLINKT(Q)
097       LD1    0,1(RLINK)     P←RLINK(P)
098       J2P    C5             如果 RTAG(Q)为 1 则转移
099       ENN2   0,2            Q←-Q
100  C6   J2NZ   C2             C6. 测试完成否?
101       LD1    8B(LLINK)      rI1←建立的第一个节点的地址
102  6H   ENT3   *              恢复变址寄存器
103  7H   ENT2   *              ▍
```

14. 命 a 是复制的非终端(运算符)节点的个数。在上边的程序中,各行的执行次数如下:064~067,1; 069,a; 070~079,$a+1$; 080~081,$n-1$; 082~088,$n-1-a$; 089~094,n; 095,$n-a$; 096~098,$n+1$; 099~100,$n-a$; 101~103,1。总的时间为$(36n+22)u$;其中大约有 20%是用来获得可用节点,40%用来遍历,以及 40%用来复制 INFO 和 LINK 信息。

15. 注释留给读者自行作出。

```
DIV  LDA    1,6
     JAZ    1F
     JMP    COPYP2
     ENTA   SLASH
     ENTX   0,6
     JMP    TREE2
     ENT6   0,1
1H   LDA    1,5
     JAZ    SUB
     JMP    COPYP2
     ST1    1F(0:2)
     ENTA   CON2
     JMP    TREE0
     ENTA   UPARROW
1H   ENTX   *
     JMP    TREE2
     ST1    1F(0:2)
     JMP    COPYP1
     ENTA   0,1
     ENT1   0,5
     JMP    MULT
     ENTX   0,1
1H   ENT1   *
     ENTA   SLASH
     JMP    TREE2
     ENT5   0,1
     JMP    SUB        ▍
```

16. 注释留给读者自行作出。

```
PWR  LDA    1,6
     JAZ    4F
     JMP    COPYP1
     ST1    R(0:2)
     LDA    0,3(TYPE)
     JANZ   2F
     LDA    1,3
     DECA   2
     JAZ    3F
     INCA   1
     STA    CON0+1
     ENTA   CON0
```

```
        JMP     TREE0
        STZ     CON0 + 1
        JMP     5F
2H      JMP     COPYP2
        ST1     1F(0:2)
        ENTA    CON1
        JMP     TREE0
1H      ENTX    *
        ENTA    MINUS
        JMP     TREE2
5H      LDX     R(0:2)
        ENTA    UPARROW
        JMP     TREE2
        ST1     R(0:2)
3H      JMP     COPYP2
        ENTA    0,1
R       ENT1    *
        JMP     MULT
        ENTA    0,6
        JMP     MULT
        ENT6    0,1
4H      LDA     1,5
        JAZ     ADD
        JMP     COPYP1
        ENTA    LOG
        JMP     TREE1
        ENTA    0,1
        ENT1    0,5
        JMP     MULT
        ST1     1F(0:2)
        JMP     COPYP1
        ST1     2F(0:2)
        JMP     COPYP2
2H      ENTX    *
        ENTA    UPARROW
        JMP     TREE2
1H      ENTX    *
        ENTA    TIMES
        JMP     TREE2
        ENT5    0,1
        JMP     ADD    ▌
```

17. 对于这样的问题的早期工作的参考文献可通过 J. Sammet 的综述性论文*CACM* **9** (1966), 555 ~ 569 找到。

18. 对于所有 j,首先置 LLINK[j]←RLINK[j]←j;使得每个节点都在长度为 1 的循环表中。然后对于 $j=n,n-1,\cdots,1$(按此次序),如果 PARENT[j] = 0 则置 r←j,否则把由 j 开始的循环表插入由 PARENT[j]开始的循环表如下:k←PARENT[j],l←RLINK[k],i←LLINK[j],LLINK[j]←k,RLINK[k]←j,LLINK[l]←i,RLINK[i]←l。这是有效的,因为(a) 每个非根节点总是由它的父亲或者由它的父亲的后裔打头;(b) 每个家族的节点以位置的次序出现在它们的父亲表中;(c)先根序是满足(a)和(b)的惟一的顺序。

20. 如果 u 是 v 的祖宗,由归纳法立即得出,在先根序下 u 居于 v 之前,而在后根序下,它接在 v 之后。反之,假定在先根序下 u 居于 v 之前,而在后根序下它接在 v 之后;我们必须证明 u 是 v 的一个祖宗。如果 u 是第一棵树的根则这是显然的。如果 u 是第一棵树的另一个节点,则 v 也必定是,因为在后根序下 u 接在 v 之后;所以归纳法适用。类似地,如果 u 不在第一棵树中,则 v 也必定不是,因为 u 在先根序下居于 v 之前。(本题也可以很容易地从习题 3 的结果得出。如果我们知道在先根序和后根序下每个节点的位置,则它为我们提供对祖宗的快速检测。)

21. 如果 NODE(P)是二元运算符,则指向它的两个操作数的指针为 P1 = LLINK(P)和 P2 = RLINK(P1) = $P。算法 D 利用了事实 P2$ = P,所以 RLINK(P1)可以改变成 Q1,即指向 NODE(P1)之导数的指针;而后来在步骤 D3 中 RLINK(P1)被复位。对于三元运算,我们将有,比如说,P1 = LLINK(P),P2 = RLINK(P1),P3 = RLINK(P2) = $P,因此难于推广二进制的这一技巧。在计算了导数 Q1 之后,我们可以暂时置 RLINK(P1)←Q1,然后在计算了下一个导数 Q2 之后,我们可以置 RLINK(Q2)←Q1 和 RLINK(P2)←Q2 以及恢复 RLINK(P1)←P2。但这肯定是不漂亮的,而且它逐渐地增长以至运算符的次数变成更高。因此,在算法 D 中暂时地改变 RLINK(P1)的设计,肯定地是一个*技巧*,而不是一项*技术*。用于控制求微分过程的更优越的方法,由于它推广到更高次数的运算符,而且不依赖于孤立的技巧,因而可以以算法 2.3.3F 为基础,参见习题 2.3.3-3。

22. 由定义立即推出,这个关系是传递的;即,如果 $T\subseteq T'$ 和 $T'\subseteq T''$,则 $T\subseteq T''$。(事实上,容易看出这个关系是一个偏序)。显然,如果我们设 f 是把诸节点变成它们自身的函数,则显

然 $l(T)\subseteq T$ 且 $r(T)\subseteq T$。因此,如果 $T\subseteq l(T')$ 或 $T\subseteq r(T')$,则我们必然有 $T\subseteq T'$。

假定 f_l 和 f_r 分别是表示 $l(T)\subseteq l(T')$ 和 $r(T)\subseteq r(T')$ 的函数。设 $f(u)=f_l(u)$ 如果 u 在 $l(T)$ 中,$f(u)=\mathrm{root}(T')$ 如果 u 是 $\mathrm{root}(T)$,否则 $f(u)=f_r(u)$。现在容易推知,f 表示 $T\subseteq T'$;例如,如果我们设 $r'(T)$ 表示 $r(T)\setminus \mathrm{root}(T)$,则我们有先根序($T$) = $\mathrm{root}(T)$ 先根序($l(T)$)先根序($r'(T)$);先根序(T') = $f(\mathrm{root}(T))$ 先根序($l(T')$)先根序($r'(T')$)。

反之则不成立,考虑图 25 中以 b 和 b' 为根的子树。

2.3.3 小节

1. 是,我们可以重新构造它们,就如同由(4)推导出(3)一样,但是要交换 LTAG 和 RTAG, LLINK 和 RLINK,而且使用队列以代替栈。

2. 在算法 F 中作下列变动:步骤 F1,变成"在先根序下森林的最后节点"。步骤 F2,在两处把"$f(x_d),\cdots,f(x_1)$"改变成"$f(x_1),\cdots,f(x_d)$"。步骤 F4,"如果 P 是先根序下的第一个节点,则终止本算法。(于是从顶到底,栈含有 $f(\mathrm{root}(T_1)),\cdots,f(\mathrm{root}(T_m))$,其中 $T_1,\cdots,T_m$ 是从左到右,给定的森林的树。)否则置 P 成为在先根序下它的前驱(在给定的表示下 P←P − c),并返回到 F2。"

3. 在步骤 D1 中,也置 S←Λ。(S 是链接到栈顶的链接变量。)步骤 D2,比如说变成"如果 NODE(P)表示一元运算符,则置 Q←S,S←RLINK(Q),P1←LLINK(P);如果它表示二元运算符,则置 Q←S,Q1←RLINK(Q),S←RLINK(Q1),P1←LLINK(P),P2←RLINK(P1)。然后实施 DIFF[TYPE(P)]"。步骤 D3 变成"置 RLINK(Q)←S,S←Q"。步骤 D4 变成"置 P←P \$"。如果我们假定 S≡LLINK(DY),则在步骤 D5 中的操作 LLINK(DY)←Q 可予以避免。这个技术显然可推广到三元的或更高阶的运算符。

4. 类似于(10)的表示用 $n-m$ 个 LLINK 和 $n+(n-m)$ 个 RLINK。在两种形式的表示之间,总的链接个数之差为 $n-2m$。当在一个节点中,LLINK 和 INFO 字段要求大约相同的空间数量以及当 m 较大时,亦即当非终端节点有较大的度数时,配置(10)就更优越。

5. 肯定地,包括穿线的 RLINK 是愚蠢的,因为 RLINK 穿线无论如何仅仅是指向 PARENT。像在 2.3.2-(4)中那样穿线的 LLINK 将是有用的,如果在树中有必要向左移动,例如,如果我们要以颠倒的后根序或以家族序来遍历一个树的话;但是这些运算如果没有穿线的 LLINK,难度也并不甚大,除非这些节点有着非常高的度数。

6. **L1**. 置 P←FIRST,FIRST←Λ。

L2. 如果 P = Λ,则终止。否则置 Q←RLINK(P)。

L3. 如果 PARENT(P) = Λ,则置 RLINK(P)←FIRST,FIRST←P;否则置 R←PARENT(P), RLINK(P)←LCHILD(R),LCHILD(R)←P。

L4. 置 P←Q 并返回到 L2。 ▌

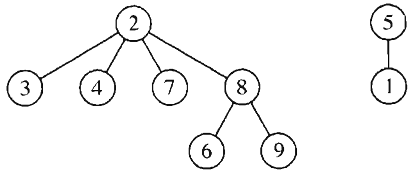
7. {1,5}{2,3,4,7}{6,8,9}。

8. 实施算法 E 的步骤 E3,然后测试是否 $j=k$。

9. PARENT[k]:5 0 2 2 0 8 2 2 8

k :1 2 3 4 5 6 7 8 9 (图见下页)

10. 一种想法是把每个根节点的 PARENT 置为它的树的节点个数之负(这些值容易保持为最新的);然后如果在步骤 E4 中 |PARENT[j]| > |PARENT[k]|,则交换 j 和 k 的作用。这个技术(由 M. D. McIlroy 给出)确保每个操作采取 $O(\log n)$ 个步骤。



为了获得更快的速度，我们可以使用 Alan Tritter 提出的下列建议：在步骤 E4 中，对于在步骤 E3 中遇到的所有值 $x \neq k$，置 PARENT[x]←k。这造成对树进行额外的扫描，但它折叠这些树，使将来的检索更快些。（见 7.4.1 小节。）

11. 只须对每个输入(P,j,Q,k)定义变换即可：

T1. 如果 PARENT(P)≠Λ，则置 j←j + DELTA(P)，P←PARENT(P)，并重复这一步。

T2. 如果 PARENT(Q)≠Λ，则置 k←k + DELTA(Q)，Q←PARENT(Q)，并重复这一步。

T3. 如果 P = Q，则校验 $j = k$（否则在这个输入中错误地包含相矛盾的等价性）。如果 P≠Q，则置 DELTA(Q)←$j - k$，PARENT(Q)←P，LBD(P)←min(LBD(P)，LBD(Q) + DELTA(Q))，UBD(P)←max(UBD(P)，UBD(Q) + DELTA(Q))。 ▌

注：在不难理解的适当条件下有可能允许“ARRAY X[l:u]”的说明与等价性混合出现，或者，允许某些变量地址在其它变量地址等价于它们之前被指定，等等。关于这个算法的进一步的发展，见*CACM* **7** (1964)，301 ~ 303，506。

12.(a)是。（如果不需要这一条件，则将有可能避免出现于步骤 A2 和 A9 中对 S 的循环。）(b)是。

13. 决定性的事实是从 P 往上引出的 UP 链，总是提出与从 Q 往上引出的 UP 链同样的变量和同样的这些变量的指数，但后一个链接可以包含具有指数 0 的变量的额外步骤。（在算法的大部分这个条件成立，除了在步骤 A9 和 A10 的执行期间外。）现在我们或从 A3 或从 A10 到达步骤 A8，而且在每种情况下已经验证了 EXP(Q)≠0。因此 EXP(P)≠0，特别是由此得出 P≠Λ，Q≠Λ，UP(P)≠Λ；UP(Q)≠Λ；现在习题中指出的结果成立。所以这个证明依赖于说明上述的 UP 链条件在算法的执行期间被保持。

16.（通过对于单棵树 T 中的节点个数使用归纳法）我们证明，如果 P 是对于 T 的一个指针，而且如果栈开始时为空，则步骤 F2 到 F4 将以栈上的单个值 $f(\mathrm{root}(T))$ 结束。对于 $n = 1$，这为真。如果 $n > 1$，有 $0 < d$ = DEGREE(root(T))个子树 $T_1, \cdots, T_d$；由归纳法和栈的性质，而且由于后根序由 $T_1, \cdots, T_d$ 后边跟随着 root(T)组成，因此如同所要求的那样，算法计算 $f(T_1), \cdots, f(T_d)$，然后 $f(\mathrm{root}(T))$。算法 F 对于森林的正确性成立。

17. **G1.** 置栈为空，并设 P 指向树的根（在后根序下的最后节点）。求值 f(NODE(P))。

G2. 把 f(NODE(P))的 DEGREE(P)个副本压入栈。

G3. 如果 P 是后根序下的第一个节点，则终止本算法。否则置 P 成为在后根序下它的前驱（这在(9)中将只不过是 P←P − c）。

G4. 利用栈顶上的值（它等于 f(NODE(PARENT(P))))求值 f(NODE(P))。把这个值弹出栈，并返回 G2。 ▌

注：类似于此的算法可以如同在习题 2 中那样以先根序为基础，而不是以后根序。事实上，家族序或层次序可予使用；在后种情况下，我们将使用队列来代替栈。

18. INFO1 和 RLINK 表格连同在正文中对计算 LTAG 的提示一起，向我们提出了以通常方式

表示的二叉树的等价者。思路是在后根序下遍历这个树，计算度数如下：

P1. 设 R，D 和 I 是栈，它们开始时为空；然后置 R⇐$n+1$，D⇐0，j←0，k←0。

P2. 如果 top(R) > $j+1$ 则转到 P5。（如果存在 LTAG 字段，我们可测试 LTAG[j] = 0 以代替 top(R) > $j+1$。）

P3. 如果 I 是空的，则终止本算法；否则置 i⇐I，k←$k+1$，INFO2[k]←INFO1[i]，DEGREE[k]⇐D。

P4. 如果 RLINK[i] = 0，则转到 P3；否则删去 R 的顶（它将等于 RLINK[i]）。

P5. 置 top(D)←top(D) + 1，j←$j+1$，I⇐j，D⇐0，而且如果 RLINK[j] ≠ 0，则置 R⇐RLINK[j]。转到 P2。 ❚

19. a) 这等价于说，SCOPE 链接彼此不相交叉。b) 森林的第一棵树包含 d_1+1 个元素。因而我们可通过归纳法进行。c) 当我们取极小值时 a)的条件保持成立。

注：由习题 2.3.2-20 得出，也可借助于反演解释 $d_1d_2\cdots d_n$：如果在后根序下的第 k 个节点是在先根序下的第 p_k 个节点，则 d_k 是出现于 $p_1p_2\cdots p_n$ 中 k 左边的大于 k 的元素的个数。

我们于其中列出森林在后根序下的每个节点的后裔个数的类似的方案，导致以下列性质作为表征的数序列 $c_1c_2\cdots c_n$：(i) $0\leqslant c_k<k$ 和(ii) $k\geqslant j\geqslant k-c_k$ 意味着 $j-c_j\geqslant k-c_k$。基于这样的序列的算法已被 J. M. Pallo 所研究，*Comp. J.* **29** (1986)，171 ~ 175。注意 c_k 是在对应的二叉树的对称序下第 k 个节点的左子树的大小。我们也可以把 d_k 解释为适当的二叉树的对称序下第 k 个节点右子树的大小，这里的二叉树即通过习题 2.3.2-5 的对偶方法对应于给定的森林的二叉树。

对于 $1\leqslant k\leqslant n$，$d_k\leqslant d'_k$ 的关系定义森林和二叉树的一个有趣的格点顺序，它首先是由 D. Tamari[Thèse(Paris：1951)]以另外一种方法介绍的；参见习题 6.2.3-32。

2.3.4.1 小节

1. (B,A,C,D,B)，(B,A,C,D,E,B)，(B,D,C,A,B)，(B,D,E,B)，(B,E,D,B)，(B,E,D,C,A,B)。

2. 设$(V_0,V_1,\cdots,V_n)$是从 V 到 V' 的最小可能长度的通路。如果现在对于某个 $j<k$，$V_j=V_k$，则$(V_0,\cdots,V_j,V_{k+1},\cdots,V_n)$是一条更短的通路。

3. (基本通路遍历 e_3 和 e_4 一次，但循环 C_2 遍历它们 -1 次，给出净总次数 0。)遍历下列诸边：$e_1,e_2,e_6,e_7,e_9,e_{10},e_{11},e_{12},e_{14}$。

4. 如果不是，则命 G'' 是通过删去使 $E_j=0$ 的每条边 e_j 所得到的 G' 的子图。于是 G'' 是没有回路的有限图，而且至少有一条边，所以由定理 A 的证明，至少有一个顶点 V，恰巧与其它的顶点 V' 相邻。设 e_j 是连接 V 到 V' 的边，则在顶点 V 处的基尔霍夫方程(1)为 $E_j=0$，与 G'' 的定义相矛盾。

5. $A=1+E_8$，$B=1+E_8-E_2$，$C=1+E_8$，$D=1+E_8-E_5$，$E=1+E_{17}-E_{21}$，$F=1+E''_{13}+E_{17}-E_{21}$，$G=1+E''_{13}$，$H=E_{17}-E_{21}$，$J=E_{17}$，$K=E''_{19}+E_{20}$，$L=E_{17}+E''_{19}+E_{20}-E_{21}$，$P=E_{17}+E_{20}-E_{21}$，$Q=E_{20}$，$R=E_{17}-E_{21}$，$S=E_{25}$。注：在这种情况下，也有可能借助于 $A,B,\cdots,S$ 来对 $E_2,E_5,\cdots,E_{25}$求解；因此有九个独立的解，说明为什么我们消去等式 1.3.3-(8)中的六个变量。

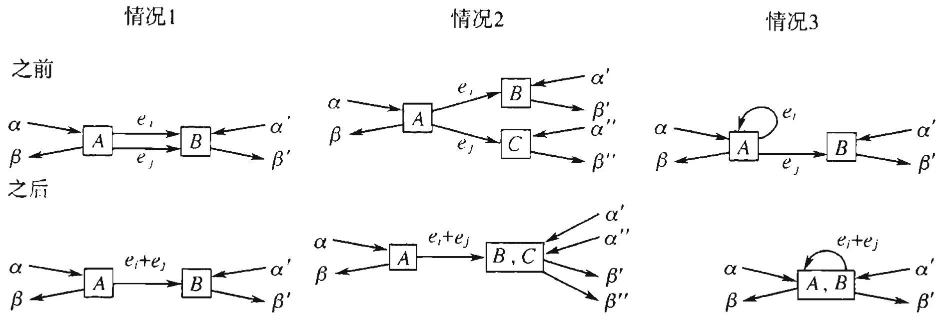
6. (下列解是基于下列思想给出的，即我们可以打印出不与前边的边构成一个回路的每条边来。)使用算法 2.3.3E，而且每对(a_i,b_i)在该算法的记号下代表 $a_i\equiv b_i$。惟一的变化是在步骤 E4 中如果 $j\neq k$ 则打印(a_i,b_i)。

为了证明这个算法是正确的,我们必须证明 (a)算法不打印出形成回路的边,以及 (b)如果 G 至少包含一个自由树,则这个算法打印出 $n-1$ 条边。如果存在从 V_j 到 V_k 的一条通路,或者如果 $j=k$,则定义 $j\equiv k$。这显然是一个等价关系,而且 $j\equiv k$ 当且仅当这个关系可从等价 $a_1\equiv b_1,\cdots,a_m\equiv b_m$ 推演出来。现在(a)成立因为算法不打印和以前已打印的边形成回路的边;(b)为真因为如果所有顶点都等价,则对恰好只有一个 k 有 PARENT[k] = 0。

然而,更有效的算法可基于深度优先查找给出。请见算法 2.3.5A 和 7.4.1 小节。

7. 基本回路:$C_0=e_0+e_1+e_4+e_9$(基本通路是 $e_1+e_4+e_9$);$C_5=e_5+e_3+e_2$;$C_6=e_6-e_2+e_4$;$C_7=e_7-e_4-e_3$;$C_8=e_8-e_9-e_4-e_3$。因此我们求得 $E_1=1,E_2=E_5-E_6,E_3=E_5-E_7-E_8,E_4=1+E_6-E_7-E_8,E_9=1-E_8$。

8. 在约化过程的每一步骤中组合两个在同一个方框开始的箭头 e_i 和 e_j,而且这足以证明这样的步骤可被逆转。于是经组合之后,我们已得到 e_i+e_j 的值,而且在组合之前,我们必须对于 e_i 和 e_j 赋以一致的值。实际上有三种不同的情况:



这里 A,B,C 代表顶点或超顶点,而诸 α 和诸 β 除 e_i+e_j 之外还代表其它给定的流量;这些流量每个可分布在若干条边当中,虽然示出的仅一条边。在情况 1 中(e_i 和 e_j 引向同一方框),我们可以任意地选择 e_i,然后 $e_j\leftarrow(e_i+e_j)-e_i$。在情况 2 中($e_i$ 和 e_j 引向不同的方框),我们必须置 $e_i\leftarrow\beta'-\alpha',e_j\leftarrow\beta''-\alpha''$。在情况 3 中($e_i$ 是一个循环而 e_j 则不是),我们必须置 $e_j\leftarrow\beta'-\alpha',e_i\leftarrow(e_i+e_j)-e_j$。在每种情况下,我们都已像所希望的那样,逆转了组合的步骤。

这一习题的结果实际上证明了,在既约的流程图中基本回路的个数,是为确定所有其它顶点流量,必须加以测定的顶点流量之极小个数。在给定的例子中,既约的流程图指出,仅有三个顶点流量(例如,a,c,d)需要加以测定,而习题 7 原来的流程图则有四个独立的边流量。在约化期间,每次情况 1 出现,我们都节省一个测定。

类似的约化过程可以以组合流入给定方框的箭头为基础,而不是以那些流出的为基础。可以证明,除了超顶点将含有不同的名称之外,这将得出相同的既约流程图。

这个习题的构造是以 Armen Nahapetian 和 F. Stevenson 的思想为基础的。关于进一步的评注,见 A. Nahapetian, *Acta Informatica* **3** (1973),37 ~ 41; D. E. Knuth 和 F. Stevenson,*BIT* **13** (1973)。313 ~ 322。

9. 从一个顶点到它本身的每一条边,变成为全部通过它自身的一条"基本回路"。如果在顶点 V 与 V' 之间有 $k+1$ 条边 $e,e',\cdots,e^{(k)}$,则做 k 条基本回路 $e'\pm e,\cdots,e^{(k)}\pm e$(按照诸边以相反或相同方向进行来选定 + 或 −),而后就像仅仅有边 e 存在那样来进行。

实际上这种情况在概念上将更为简单,如果我们以这样一种方式来定义图的话,即,在顶点

之间允许有多条边，而且允许边从一个顶点到它自身；通路和回路将借助于边而不是顶点来进行定义。事实上，这种类型的定义，已在 2.3.4.2 小节中对有向图给出。

10. 如果端点已经全部连接在一起，则对应的图必须在技术的意义下加以连接。导线的极小数显然将不含回路，所以我们必须有一个自由树。按定理 A，自由树包含 $n-1$ 条导线，而具有 n 个顶点和 $n-1$ 条边的图是一个自由树，当且仅当，它是连通的。

11. 只需证明，当 $n>1$ 且 $c(n-1,n)$ 是 $c(i,n)$ 之极小者时，至少存在一个极小代价的树，其中 T_{n-1} 是连线到 T_n 的。（因为，任何具有 $n>1$ 个接线端和 T_{n-1} 连线到 T_n 的极小代价树，必须也是有 $n-1$ 个接线端的极小代价树，如果我们利用算法中所指出的约定，认为 T_{n-1} 和 T_n 是"公共的"的话。）

为证明上述命题，假设我们有一个极小代价树，其中 T_{n-1} 不连线到 T_n。如果我们加上导线 T_{n-1}—T_n，则我们就得到一条回路，而且在该回路中的任何其它导线均可去掉；把接触 T_n' 的另一条导线去掉，给了我们另一棵树，它的总代价不大于原来的，而且在该树中出现 T_{n-1}—T_n。

12. 对于 $1\leqslant i<n$，保持两个辅助表格 $a(i)$ 和 $b(i)$，表示这样的事实，即，从 T_i 到选定的端点的代价最小的连接是到 $T_{b(i)}$，而且它的代价是 $a(i)$；开始时 $a(i)=c(i,n)$ 和 $b(i)=n$。然后进行下列操作 $n-1$ 次：求 i，使得 $a(i)=\min_{1\leqslant j<n}a(j)$；连接 T_i 到 $T_{b(i)}$；对于 $1\leqslant j<n$，如果 $c(i,j)<a(j)$，则置 $a(j)\leftarrow c(i,j)$ 和 $b(j)\leftarrow i$；并且置 $a(i)\leftarrow\infty$。这里的 $c(i,j)$ 当 $j<i$ 时就是 $c(j,i)$。

（避免使用 ∞，代之以保持尚未选定的那些 j 的单路链接表，是较为有效的。采用或不采用这个直截了当的改进，这一算法均花去 $O(n^2)$ 个操作。）见 E. W. Dijkstra, *Proc. Nederl. Akad. Wetensch.* **A63** (1960), 196～199; D. E. Knuth, *Stanford GraphBase* (New York: ACM Press, 1993), 460～497。在 7.5.4 小节中讨论了求一个极小代价跨越树的好得多的算法。

13. 如果对于某 $i\neq j$，没有从 V_i 到 V_j 的回路，则没有转置之乘积将把 i 移到 j。所以如果生成了所有的排列，则这图必连通。反过来，如果它连通，则必要时撤去些边直到我们得到一个树为止。然后重新将顶点编号使得 V_n 仅与另一个顶点相邻，称之为 V_{n-1}。（见定理 A 的证明。）现在不同于 $(n-1\ n)$ 的转置，形成一个具有 $n-1$ 个顶点的自由树；所以由归纳法，如果 π 是 $\{1,2,\cdots,n\}$ 上的任何排列，它使 n 保持固定，则 π 可以写成那些转置的乘积。如果 π 把 n 移到 j，则 $\pi(j\ n-1)(n-1\ n)=\rho$ 固定 n；因此 $\pi=\rho(n-1\ n)(j\ n-1)$ 可以写成为给定转置之一乘积。

2.3.4.2 小节

1. 设 $(e_1,\cdots,e_n)$ 是一条从 V 到 V' 的最小长度的有向通路。现在如果对于 $j<k$，$\mathrm{init}(e_j)=\mathrm{init}(e_k)$，则 $(e_1,\cdots,e_{j-1},e_k,\cdots,e_n)$ 将是更短的通路；如果对于 $j<k$，$\mathrm{fin}(e_j)=\mathrm{fin}(e_k)$，则可应用同样的论证。因此 $(e_1,\cdots,e_n)$ 是简单的。

2. 其中所有符号皆相同的那些回路：C_0, C_8, C''_{13}, C_{17}, C''_{19}, C_{20}。

3. 例如，使用三个顶点 A, B, C，连同从 A 到 B 和从 A 到 C 的有向边。

4. 如果没有有向回路，则算法 2.2.3T 把 G 拓扑排序。如果有一个有向回路，则拓扑排序显然是不可能的。（取决于对这道习题如何解释，长度为 1 的有向回路可从上述讨论中排斥出去。）

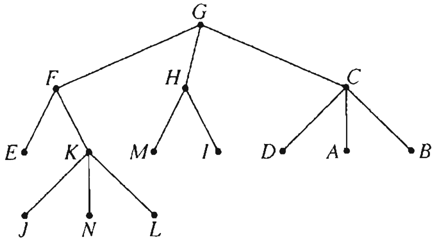
5. 设 k 是最小整数，使得对于某 $j\leqslant k$，$\mathrm{fin}(e_k)=\mathrm{init}(e_j)$。则 $(e_j,\cdots,e_k)$ 是一条有向回路。

6. 假的（从技术上来说），仅仅因为从一个顶点到另一个顶点可能有若干条不同的有向边。

7. 对于有限的有向图为真：因为如果我们在任何顶点 V 开始并且沿着惟一可能的有向通路，则绝不会遇到任何顶点两次，所以必然最终达到顶点 R（惟一无后继的顶点）。对于无限的有

向图,这一结果显然是假的,因为我们可以有顶点 $R, V_1, V_2, V_3, \cdots$ 以及对于 $j \geq 1$,从 V_j 到 V_{j+1} 的有向边。

9. 所有的有向边都向上指。



10. **G1**. 置 $k \leftarrow P[j], P[j] \leftarrow 0$。

G2. 如果 $k = 0$,则停止;否则置 $m \leftarrow P[k], P[k] \leftarrow j, j \leftarrow k, k \leftarrow m$,而且重复步骤 G2。 ▍

11. 这个算法把算法 2.3.3E 与上题的方法结合起来,使得所有有向树的有向边对应于有向图中实际的有向边;$S[j]$ 是一份辅助表格,它告知一条有向边是从 j 到 $P[j]$($S[j] = +1$)还是从 $P[j]$ 到 j($S[j] = -1$)。开始时,$P[1] = \cdots = P[n] = 0$。下列诸步骤可用来处理每条有向边 (a, b):

C1. 置 $j \leftarrow a,\ k \leftarrow P[j],\ P[j] \leftarrow 0,\ s \leftarrow S[j]$。

C2. 如果 $k = 0$,则转到 C3;否则置 $m \leftarrow P[k], t \leftarrow S[k], P[k] \leftarrow j, S[k] \leftarrow -s, s \leftarrow t, j \leftarrow k, k \leftarrow m$,而且重复步骤 C2。

C3. (现在 a 表现为它的树的根。)置 $j \leftarrow b$,然后,如果 $P[j] \neq 0$,则重复地置 $j \leftarrow P[j]$ 直到 $P[j] = 0$ 为止。

C4. 如果 $j = a$,则转到 C5;否则置 $P[a] \leftarrow b, S[a] \leftarrow +1$,打印 (a, b) 以作为一条属于自由子树的有向边,而且终止。

C5. 打印"CYCLE",后面紧跟"(a, b)"。

C6. 如果 $P[b] = 0$,则终止。否则如果 $S[b] = +1$,则打印"$+(b, P[b])$",否则打印"$-(P[b], b)$";置 $b \leftarrow P[b]$ 并重复步骤 C6。 ▍

注:如果把在答案 2.3.3-10 中 McIlroy 的建议加入进来,这个算法至多将花费 $O(m \log n)$ 步,但是还有只需 $O(m)$ 步的好得多的算法:使用深度优先查找来构造"棕榈树",连同对于每个"棕榈复叶"的基本回路[R. E. Tarjan, *SICOMP* **1** (1972), 146 ~ 150]。

12. 它等于入度;每个顶点的出度仅可为 0 或 1。

13. 定义 G 之有向子树的序列如下:G_0 仅是顶点 R。G_{k+1} 是 G_k 加上 G 的任何不在 G_k 中但对于它有一条从 V 到 V' 的有向边的任何顶点 V,其中 V' 在 G_k 中,再加上对于每个这样的顶点的一条这样的有向边 $e[V]$。由归纳法立即得知,对所有 $k \geq 0$,G_k 是有向树,而且如果在 G 中从 V 到 R 有一长度为 k 的有向通路,则 V 在 G_k 中。因此 G_∞,即在任何 G_k 中所有的 V 和 $e[V]$ 之集合,就是所求的 G 的有向子树。

14. 在字典序下,

$(e_{12}, e_{20}, e_{00}, e'_{01}, e_{10}, e_{01}, e'_{12}, e_{22}, e_{21})$, $(e_{12}, e_{20}, e_{00}, e'_{01}, e'_{12}, e_{22}, e_{21}, e_{10}, e_{01})$,

$(e_{12}, e_{20}, e'_{01}, e_{10}, e_{00}, e_{01}, e'_{12}, e_{22}, e_{21})$, $(e_{12}, e_{20}, e'_{01}, e'_{12}, e_{22}, e_{21}, e_{10}, e_{00}, e_{01})$,

$(e_{12}, e_{22}, e_{20}, e_{00}, e'_{01}, e_{10}, e_{01}, e'_{12}, e_{21})$, $(e_{12}, e_{22}, e_{20}, e_{00}, e'_{01}, e'_{12}, e_{21}, e_{10}, e_{01})$,

$(e_{12}, e_{22}, e_{20}, e'_{01}, e_{10}, e_{00}, e_{01}, e'_{12}, e_{21})$, $(e_{12}, e_{22}, e_{22}, e'_{01}, e'_{12}, e_{21}, e_{10}, e_{00}, e_{01})$
这八种可能性,来自于 e_{00}或 e'_{01}, e_{10}或 e'_{12}, e_{21}或 e_{22}中,哪个应当排在其它之前的独立选择。

15. 对有限图正确:如果它是连通和平衡的,并且有一个以上的顶点,它有与所有顶点都接触的欧拉路径。(但通常是错误的。)

16. 考虑具有顶点 $V_1, \cdots, V_{13}$,和对于在叠 j 中的每个 k 具有一条从 V_j 到 V_k 的有向边的有向图 G。赢得游戏,就等价于在 G 中寻找出一条欧拉路径,因为当第 4 条指向 V_{13}的有向边接触到(也就是说第 4 张 K 被翻开)时,游戏结束。现在如果这局游戏得胜,则由引理 E,所述图是有向树。反之,如果所述图是有向树,则由定理 D 游戏获胜。

17. $\frac{1}{13}$。可以得到这个答案,如同作者第一次得到它那样,办法是以 2.3.4.4 小节的方法为基础,通过对特殊类型的有向树进行吃力的枚举,而且应用生成函数,等等。从以下简单而直接的证明也容易得出:即定义翻转整副所有牌的次序如下:遵照游戏的规则直到停止为止,然后通过翻转第一张可利用的牌来“作弊”(找不空的第一叠,从叠 1 顺时针方向前进)并与以前一样继续下去,直到最终把所有的牌都翻转了为止。在翻转的次序下,诸牌处于完全随机的次序(因为一张牌的值,直到翻转它之前无须确定)。所以问题恰恰在计算,在一副随机洗过的牌中的最后一张牌是 K 的概率。更一般地说,当游戏玩完时,k 张牌仍然朝下的概率,是在随机的洗牌下最后的 K 后面跟着 k 张牌的概率,这就是$4!\binom{51-k}{3}\frac{48!}{52!}$。因此玩这种游戏的人,如果不作弊,则在每一局游戏中,平均恰恰翻转 42.4 张牌。注:类似地可以证明,游戏者在上边描述的过程中将需要“作弊”k 次的概率,恰恰是由斯特林数$\left[{13 \atop k+1}\right]\Big/13!$ 给出的。(见等式 1.2.10-(9)和习题 1.2.10-7;在习题 1.2.10-18 中考虑了更一般的纸牌的情况。)

18. a)如果有一条回路$(V_0, V_1, \cdots, V_k)$,其中必须有 $3 \leqslant k \leqslant n$,则对应于这条回路的 k 条边的 A 的 k 行之和,带上相应的符号,是一全 0 的行;所以如果 G 不是自由树,则 A_0 的行列式为 0。

但是如果 G 是自由树,则我们可以认为它是具有根 V_0 的有序树,而且我们可以重新排列 A_0 的行与列,使得这些列处于先根序之下,并且使得第 k 行对应于从第 k 个顶点(列)到它的父亲的边;于是这个矩阵是在对角线上为 ± 1 的三角矩阵,所以行列式为 ± 1。

b)由 Binet-Cauchy 公式(习题 1.2.3-46),我们有

$$\det A_0^{\mathrm{T}} A_0 = \sum_{1 \leqslant i_1 < \cdots < i_n \leqslant m} (\det A_{i_1 \cdots i_n})^2$$

其中 $A_{i_1 \cdots i_n}$表示由 A_0 的行 $i_1, \cdots, i_n$ 组成的矩阵(于是对应于 G 的 n 条边的一种选择)。现在由 a)得出结果。

[请见 S. Okada 和 R. Onodera 的论文,*Bull. Yamagata Univ.* **2** (1952),89~117。]

19. a) $a_{00}=0$ 和 $a_{jj}=1$ 的条件,恰是有向树之定义的条件 a),b)。如果 G 不是有向树,则有一个有向回路(由习题 7),而且,对应于该有向回路中之顶点的 A_0 的诸行,将加成为一全 0 的行;因此 $\det A_0=0$。如果 G 是有向树,则对于每个家庭的诸儿子指定一个任意的顺序,并把 G 看成是有序树。现在排列 A_0 的行与列,直到它们对应于顶点的先根序为止。由于应用于列的同样排列已经应用于行,这行列式不变;而且得到的矩阵是在每一对角线位置上有 $+1$ 的三角矩阵。

b)我们可以假定对于所有的 j, $a_{0j}=0$,因为没有从 V_0 发出的边能加入到有向子树中。我们还可以假定对于所有的 $j \geqslant 1$, $a_{jj}>0$,因为否则整个的第 j 行为 0,而且显然地没有有向子树。现在对有向边数用归纳法:如果 $a_{jj}>1$,则设 e 是从 V_j 引出的某条有向边;设 B_0 是一个类似 A_0 的

矩阵，但删去了有向边 e；且设 C_0 为类似 A_0 的矩阵，但删去了除从 V_j 引出的 e 之外的所有有向边。例：如果 $A_0 = \begin{pmatrix} 3 & -2 \\ -1 & 2 \end{pmatrix}$，$j = 1$，$e$ 是从 V_1 到 V_0 的有向边，则 $B_0 = \begin{pmatrix} 2 & -2 \\ -1 & 2 \end{pmatrix}$，$C_0 = \begin{pmatrix} 1 & 0 \\ -1 & 2 \end{pmatrix}$。一般说来，我们有 $\det A_0 = \det B_0 + \det C_0$，因为这些矩阵在除了 j 行之外的所有行上都一致，而且在 j 行上 A_0 为 B_0 和 C_0 之和。而且，G 之有向子树的个数，是不使用 e 的子树的个数（即 $\det B_0$，由归纳法）加上使用 e 的子树的个数（即 $\det C_0$）。

注：通过和偏微分方程理论中相似概念的类比，通常把矩阵 A 称做图的*拉普拉斯矩阵*。如果我们从矩阵 A 中删去行的任何集合 S，以及列的同样集合，得到的矩阵的行列式是根是顶点 $\{V_k \mid k \in S\}$ 且有向边属于给定的有向图的有向森林的个数。对于有向树的矩阵树定理是由 J. J. Sylvester 未加证明地于 1857 年提出的（请见习题 28 答案），而后它被忘却了许多年，一直到由 W. T. Tuttle 独立地重新发现它为止［*Proc. Cambridge Phil. Soc.* **44** (1948)，463～482，§3］。当矩阵 A 是对称的时，在*无向图*的特殊情况下，由 C. W. Borchardt 给出了第一个公开的证明［*Crelle* **57** (1860)，111～121］。许多作者把这个定理归功于基尔霍夫，但基尔霍夫证明的是一个十分不同的（尽管有关）结果。

20. 利用习题 18，我们求得 $B = A_0^{\mathrm{T}} A_0$。或者，利用习题 19，B 是以两条有向边代替 G 的每条边（在每一方向各一条）得到的有向图 G' 的矩阵 A_0；而且 G 的每个自由子树，惟一地对应于 G' 的以 V_0 为根的有向子树，因为有向边的方向是由根的选择确定的。

21. 如同在习题 19 中那样构造矩阵 A 和 A^*。对于在图 36 和图 37 中的例图 G 和 G^*，

$$A = \begin{pmatrix} 2 & -2 & 0 \\ -1 & 3 & -2 \\ -1 & -1 & 2 \end{pmatrix}, \quad A^* = \begin{array}{c} \\ [00] \\ [10] \\ [20] \\ [01] \\ [01] \\ [21] \\ [12] \\ [12] \\ [22] \end{array} \begin{array}{c} \begin{array}{ccccccccc} [00] & [10] & [20] & [01] & [01] & [21] & [12] & [12] & [22] \end{array} \\ \left(\begin{array}{ccccccccc} 2 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & -1 & -1 & 0 & 0 & 0 & 0 \\ \hline 0 & -1 & 0 & 3 & 0 & 0 & -1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 3 & 0 & -1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 3 & -1 & -1 & 0 \\ \hline 0 & 0 & -1 & 0 & 0 & -1 & 3 & 0 & -1 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & 3 & -1 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 2 \end{array}\right) \end{array}$$

把不确定的 λ 加到 A 和 A^* 的每个对角的元素上。如果 $t(G)$ 和 $t(G^*)$ 是 G 和 G^* 的有向子树的个数，则我们有 $\det A = \lambda t(G) + O(\lambda^2)$，$\det A^* = \lambda t(G^*) + O(\lambda^2)$。（由习题 22，一个平衡图的有向子树的个数，对于任何给定的根是相同的，但我们不需要这一事实。）

如果我们把顶点 V_{jk} 对相等的 k 分组，则矩阵 A^* 可如上所示划分。设对于所有的 j 和 k，$B_{kk'}$ 是由 V_{jk} 的行和 $V_{j'k'}$ 的列所组成的 A^* 的子矩阵，使得 V_{jk} 和 $V_{j'k'}$ 都在 G^* 中。通过把每个子矩阵的第 2，…，第 m 列加到第一列，然后从每个子矩阵的第 2，…，第 m 行减去第一行，矩阵 A^* 被变换成使得

$$B_{kk'} = \begin{pmatrix} a_{kk'} & * & \cdots & * \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \text{对于 } k \neq k', \quad B_{kk} = \begin{pmatrix} \lambda + a_{kk} & * & \cdots & * \\ 0 & \lambda + m & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda + m \end{pmatrix}$$

在转置的子矩阵的顶行上的星号是无关的，因为 A^* 的行列式现在可见为 $(\lambda + m)^{(m-1)n}$ 乘

$$\det\begin{pmatrix} \lambda + a_{00} & a_{01} & \cdots & a_{0(n-1)} \\ a_{10} & \lambda + a_{11} & \cdots & a_{1(n-1)} \\ \vdots & & \ddots & \vdots \\ a_{(n-1)0} & a_{(n-1)1} & \cdots & \lambda + a_{(n-1)(n-1)} \end{pmatrix} = \lambda t(G) + O(\lambda^2)$$

注意当 $n=1$ 且从 V_0 到它自身有 m 条有向边时,我们特别发现在 m 个标记的节点上可能恰有 m^{m-1}棵有向树。2.3.4.4 小节将以非常不同的方法得到这一结果。

当 G 是一个任意的有向图时,这个推导可以推广来确定 G^* 的有向子树的个数;请见 R. Dawson 和 I.J.Good,*Ann. Math. Stat.* **28**(1957),946~956; D. E. Knuth,*Journal of Combinatorial Theory* **3** (1967),309~314。另一个选择,即纯粹组合的证明,已由 J. B. Orlin 给出 ,*Journal of Combinatorial Theory* **B25** (1978),187~198。

22. 总数是$(\sigma_1+\cdots+\sigma_n)$乘以由给定的边 e_1 开始的欧拉回路的条数,其中 $\mathrm{init}(e_1)=V_1$。由引理 E,每条这样的回路,确定以 V_1 为根的有向子树,而且对于 T 个有向子树中的每一个,有 $\prod_{j=1}^{n}(\sigma_j-1)!$条满足定理 D 的三个条件的通路,对应于进入 P 的有向边$\{e \mid \mathrm{init}(e)=V_j, e\neq e[V_j], e\neq e_1\}$的不同顺序。(习题 14 提供了一个简单的例子。)

23. 像在提示中那样构造具有 m^{k-1}个顶点的有向图 G_k,而且以$[x_1,\cdots,x_k]$表示那里提及的有向边。对于每一个有着极大周期长度的函数,通过令 $f(x_1,\cdots,x_k)=x_{k+1}$,如果有向边$[x_1,\cdots,x_k]$的后边接有$[x_2,\cdots,x_{k+1}]$的话,我们可以定义一条惟一对应的欧拉回路。(如果一条欧拉回路仅仅是另一条的循环排列,则我们认为它们是相同的。)现在在习题 21 的意义下 $G_k=G_{k-1}^*$,所以 G_k 的有向子树的个数是 G_{k-1}的有向子树个数的 $m^{m^{k-1}-m^{k-2}}$倍;由归纳法,G_k 有 $m^{m^{k-1}-1}$个有向子树,而且有 $m^{m^{k-1}-k}$个有给定的根。因此,由习题 22,有极大周期的函数的个数,即以给定的有向边开始的 G_k 的欧拉回路的条数,是 $m^{-k}(m!)^{m^{k-1}}$。[对于 $m=2$,这个结果属于 C. Flye Sainte-Marie,*L'Intermédiaire des Mathématiciens* **1** (1894),107~110。]

24. 对于$0\leqslant j\leqslant m$,定义一个新的有 E_j 个 e_j 的副本的有向图。这个图是平衡的,因此由定理 G 它包含一条欧拉回路$(e_0,\cdots)$。通过从这条欧拉回路删去边 e_0,就得到所求的通路。

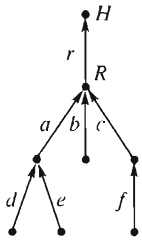
25. 对于在集合 $I_j=\{e \mid \mathrm{init}(e)=V_j\}$和 $F_j=\{e \mid \mathrm{fin}(e)=V_j\}$中的所有有向边,指定任意的顺序。对于 I_j 中的每条有向边 e,如果在 I_j 的顺序下 e'在 e 之后,则令 ATAG$(e)=0$,ALINK$(e)=e'$;并且如果 e 在 I_j 中是最后一个而 e' 在 F_j 中是第一个,则令 ATAG$(e)=1$ 以及 ALINK$(e)=e'$。在后一种情况下,如果 F_j 为空,则令 ALINK$(e)=\Lambda$。颠倒 init 和 fin 的作用,通过相同的规则,定义 BLINK 和 BTAG。

例(在每个有向边的集合中使用字符顺序)见下页图。

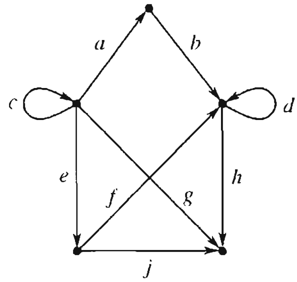
注:如果在该有向树表示中,我们加上从 H 到它自己的另一条有向边,就得到一种有趣的情况:或者是得到在表头处互相交换 LLINK,LTAG,RLINK,RTAG 的标准约定 2.3.1-(8),或者(如果新的有向边在这顺序下排在最后)是得到在与树的根相关联的节点中的标准约定,但 RTAG=0除外。

本习题是基于 W. C. Lynch 同作者交流的想法给出的。利用这一表示,像算法 2.3.1S 这样的树遍历算法能否被推广到不是有向树的有向图的类中?

27. 令 a_{ij}是对从 V_i 到 V_j 的所有有向边 e 的 $p(e)$之和。我们要证明,对于所有 j,$t_j=\sum_i a_{ij}t_i$。由于$\sum_i a_{ji}=1$,我们必须证明$\sum_i a_{ji}t_j=\sum_i a_{ij}t_i$。但这并不难,因为等式两边表示取遍 G 的子图$\{e_1,\cdots,e_n\}$的所有乘积 $p(e_1)\cdots p(e_n)$之和,使得 $\mathrm{init}(e_i)=V_i$,而且使得有惟一的有向回路被包



有向边	ALINK	ATAG	BLINK	BTAG
a	d	1	b	0
b	Λ	1	c	0
c	f	1	r	1
d	Λ	1	e	0
e	Λ	1	a	1
f	Λ	1	c	1
r	a	1	Λ	1



有向边	ALINK	ATAG	BLINK	BTAG
a	c	0	b	1
b	a	1	d	0
c	e	0	a	1
d	h	0	f	0
e	g	0	f	1
f	j	0	d	1
g	c	1	h	0
h	b	1	j	0
j	e	1	Λ	1

含在 $\{e_1,\cdots,e_n\}$ 当中,而在这个回路中包含 V_j。删去这个回路中的任何一条有向边产生一个有向树;通过提取离开 V_j 的有向边的因子就得到等式的左边,而右边对应于进入 V_j 的那些边。

在某种意义下,本题是习题 19 和 26 的组合。

28. 在展开式中的每一项是 $a_{1p_1}\cdots a_{mp_m}b_{1q_1}\cdots b_{nq_n}$,其中对于 $1\leqslant i\leqslant m$,$0\leqslant p_i\leqslant n$,对于 $1\leqslant j\leqslant n$,$0\leqslant q_j\leqslant m$,乘以某个整系数。把这个乘积表示为在顶点 $\{0,u_1,\cdots,u_m,v_1,\cdots,v_n\}$ 上的有向图,连同从 u_i 到 v_{p_i} 以及从 v_j 到 u_{q_j} 的有向边,其中 $u_0=v_0=0$。

如果这个有向图包含一个回路,则整系数为零。对于对应于形如

$$a_{i_0j_0}b_{j_0i_1}a_{i_1j_1}\cdots a_{i_{k-1}j_{k-1}}b_{j_{k-1}i_0} \tag{$*$}$$

的因子的每个回路,其中的下标 $(i_0,i_1,\cdots,i_{k-1})$ 都是不同的,下标 $(j_0,j_1,\cdots,j_{k-1})$ 也不同。包含 $(*)$ 作为因子的所有项之和是对于 $0\leqslant l<k$,对 $0\leqslant j\leqslant n$ 置 $a_{i_lj}\leftarrow[j=j_l]$,对 $0\leqslant i\leqslant m$ 置 $b_{j_li}\leftarrow[i=i_{(l+1)\bmod k}]$,在其它的 $m+n-2k$ 行中保持变量不变得到的行列式与 $(*)$ 的积。这个行列式恒等于零,因为在顶部的行 $i_0,i_1,\cdots,i_{k-1}$ 之和等于在底部的行 $j_0,j_1,\cdots,j_{k-1}$ 之和。

另一方面,如果有向图不含回路,则整系数是 $+1$。这是因为每个因子 a_{ip_i} 和 b_{jq_j} 必定来自行列式的对角线:如果任何非对角线元素 $a_{i_0j_0}$ 是在顶部的行 i_0 选定的,我们必须从底部的行 j_0 选择某个非对角线元素 $b_{j_0i_1}$,因此必须从顶部的行 i_1 选择某个非对角线元素 $a_{i_1j_1}$,等等,因而迫使形成一个循环。

这样一来系数是 $+1$,当且仅当对应的有向图是有根 0 的有向树。这样的项的个数(因此这样的有向树的个数)通过置每个 a_{ij} 和 b_{ji} 为 1 而得到;例如,

$$\det\begin{pmatrix}4&0&1&1&1\\0&4&1&1&1\\1&1&3&0&0\\1&1&0&3&0\\1&1&0&0&3\end{pmatrix}=\det\begin{pmatrix}4&0&1&1&1\\-4&4&0&0&0\\1&1&3&0&0\\0&0&-3&3&0\\0&0&-3&0&3\end{pmatrix}=\det\begin{pmatrix}4&0&3&1&1\\0&4&0&0&0\\2&1&3&0&0\\0&0&0&3&0\\0&0&0&0&3\end{pmatrix}=$$

$$\det\begin{pmatrix}4&3\\2&3\end{pmatrix}\cdot4\cdot3\cdot3$$

一般说来,我们得到 $\det\begin{pmatrix} n+1 & n \\ m & m+1 \end{pmatrix}\cdot(n+1)^{m-1}\cdot(m+1)^{n-1}$。

注:J. J. Sylvester 在*Quarterly J. of Pure and Applied Math*. **1** (1857),42 ~ 56 中,考虑了 $m = n$ 和 $a_{10} = a_{20} = \cdots = a_{m0} = 0$ 的特殊情况,其中他(正确地)猜想,项的总数是 $n^n(n+1)^{n-1}$。他也未加证明地指出当 $a_{ij} = \delta_{ij}$时$(n+1)^{n-1}$个非零的项都对应于在$\{0,1,\cdots,n\}$上所有连通的无回路图。在该特殊情况下,他把行列式归约成为在习题 19 的矩阵树定理下的形式,例如,

$$\det\begin{pmatrix} b_{10}+b_{12}+b_{13} & -b_{12} & -b_{13} \\ -b_{21} & b_{20}+b_{21}+b_{23} & -b_{23} \\ -b_{31} & -b_{32} & b_{30}+b_{31}+b_{32} \end{pmatrix}$$

Cayley 在*Crelle* **52** (1856),279 中引述了这个结果,并把它归功于 Sylvester;具有讽刺性的是,关于这样的图的个数的定理通常归功于 Cayley。

把给定行列式的前 m 行变负,然后把前 m 列变负,我们可以把这个习题归结为矩阵树定理。

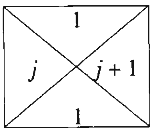
[有着在本习题中考虑的一般形式的矩阵在求解偏微分方程的迭代方法中是重要的,因此说它们具有"性质 $\mathscr{A}$"。比如说,请见 Louis A. Hageman 及 David. M. Young, *Applied Iterative Methods* (Academic Press, 1981),第 9 章。]

2.3.4.3 小节

1. 根是空的序列;有向边从$(x_1,\cdots,x_n)$到$(x_1,\cdots,x_{n-1})$。

2. 取一个四位一体型,并把它转动 180°以获得另一个四位一体型;这两个式样通过 2 × 2 型的复制,以一种显然的方法来铺平面(不作进一步的转动)。

3. 对于所有的正整数 j 考虑四位一体型



的集合。则右半平面可以以不可数多的方法来平铺;但是,放在该平面之中心的无论是什么方块,其可以向左继续的距离都有了有限的极限。

4. 对于 $n = 1, 2, \cdots$,系统地枚举平铺一 $n\times n$ 方块的所有可能方法,寻求在这些方块之内的环形解。如果没有办法平铺这个平面,则无限性引理告诉我们,有一个整数 n 没有 $n\times n$ 解。如果有一种办法来铺这个平面,则这假定告诉我们,存在一个 n,具有包含着产生一环形解的矩形的 $n\times n$ 解。因此,在两种情况下的每一种,这算法都将终止。(但正像下一习题所示,所述的假定是假的,而且事实上没有这样一种算法,它在有限步之内将确定是否存在一个方法,以给定的式样集合来铺这平面。)

5. 通过注意这样一点开始,即在任何一个解中,我们需要以 2 × 2 组所复制的种类$\begin{smallmatrix}\alpha & \beta \\ \gamma & \delta\end{smallmatrix}$。然后,步骤 1:仅仅考虑 α 正方形,证明$\begin{smallmatrix}a & b \\ c & d\end{smallmatrix}$型式必须以$\alpha$ 个正方形的 2 × 2 组来复制。步骤 $n > 1$:确定必然出现在一高度和宽度为 $2^n - 1$ 之十字形区域里的模式。这些十字型的中间,有遍布整个

平面复制的模式 $\begin{matrix} Na & Nb \\ Nc & Nd \end{matrix}$。

例如，在步骤 3 之后，我们将知道整个平面上 7×7 方块的内容，以单位长的小分条分开，每个是 8 个单位。在中心的类 Na 的 7×7 方块，有下列形式：

αa	βKQ	αb	βQP	αa	βBK	αb
γPJ	δNa	γRB	δQK	γLJ	δNb	γPB
αc	βDS	αd	βQTY	αc	βBS	αd
γPQ	δPJ	γPXB	δNa	γPQ	δRB	γRB
αa	βUK	αb	βDP	αa	βBK	αb
γTJ	δNc	γSB	δDS	γSJ	δNd	γTB
αc	βQS	αd	βDT	αc	βBS	αd

中间的列和中间的行，是刚刚在步骤 3 期间填入的“十字”；其它四个 3×3 正方形在步骤 2 之后填入；这些恰好在 7×7 正方形的右边和下边的正方形，是在步骤 4 时有待填入的 15×15 十字的一部分。

关于导致仅有 35 个四位一体型之集合的类似的构造，只不过是有非环形解的，见 R. M. Robinson, *Inventiones Math*. **12** (1971), 177 ~ 209。Robinson 也揭示了一组六个多面体的形式，它仅仅非环形地铺平面，甚至当允许转动和反射时也是。1974 年，Roger Penrose 基于黄金分割而不是正方形栅格，发现了仅有两个多边形的集合，它们仅仅非周期地铺平面；这导致对于仅仅非环形的解的只有 16 种四位一体型的一个集合[请见 B. Grünbaum 和 G. C. Shephard, *Tilings and Patterns* (Freeman, 1987)，第 10 ~ 11 章；Martin Gardner, *Penrose Tiles to Trapdoor Ciphers* (Freeman, 1989)，第 1 ~ 2 章]。

6. 设 k 和 m 是固定的。考虑一个有向树，它的每个顶点，对于某个 n，表示把 $\{1,\cdots,n\}$ 分划成不包含长度为 m 之算术级数的 k 个部分之一分划。分划 $\{1,\cdots,n+1\}$ 的一个节点，是分划 $\{1,\cdots,n\}$ 的一个节点的儿子，如果这两个分划在 $\{1,\cdots,n\}$ 上一致的话。如果有通向根的一条无限通路，则我们将有一种方法来把所有整数分划成没有长度为 m 之算术级数的 k 个集合。因此，由无限性引理和van der Waerden 定理，这个树是有限的。(如果 $k=2$, $m=3$，则这个树可以很快地用手算来计算，而且 N 的最小值为 9。见 *Studies in Pure Mathematics*, L. Mirsky 编 (Academic Press, 1971), 251 ~ 260，其中有关于 van der Waerden 怎样发现他的定理之证明的有趣说明。)

7. 正整数可以被分划成两个集合 S_0 和 S_1，使得其中任何一个都不包含任何无限的可计算的序列(参见习题 3.5-32)。所以，特别地没有无限的算术级数。由于没有办法把部分解置入其每个顶点均有有限度数的树中，因此不能应用定理 K。

8. 令一“反例序列”是违背 Kruskal 定理的树的无限序列，如果存在这样的序列的话。假定定理为假；则令 T_1 是具有最小可能的节点数的树，它使得 T_1 可以成为一反例序列中的第一个树；如果已经选定了 $T_1,\cdots,T_j$，则命 T_{j+1} 是使得 $T_1,\cdots,T_j,T_{j+1}$ 成为一反例序列前面的具有最小可能的节点数的树。这一过程定义了一反例序列 $\langle T_n\rangle$。这些 T 中都不仅仅有一个根而已。现在我们非常仔细地来考虑这个序列：

(a) 假设有子序列 $T_{n_1}, T_{n_2},\cdots$，对于它 $l(T_{n_1}), l(T_{n_2}),\cdots$ 是一个反例序列。这是不可能的，因为否则 $T_1,\cdots,T_{n_1-1}, l(T_{n_1}), l(T_{n_2}),\cdots$ 将是一反例序列，与 T_{n_1} 的定义相矛盾。

(b)由于(a),仅仅有有限多的 j,对于它说来,对任何 $k>j$, $l(T_j)$不能被嵌入 $l(T_k)$中。因此,通过取 n_1 大于任何这样的 j,我们可以找到一个子序列,使得 $l(T_{n_1})\subseteq l(T_{n_2})\subseteq l(T_{n_3})\subseteq\cdots$。

(c)现在由习题2.3.2-22的结果,对于任何 $k>j$, $r(T_{n_j})$不可能嵌入 $l(T_{n_k})$中,否则 $T_{n_j}\subseteq T_{n_k}$。因此 $T_1,\cdots,T_{n_1-1},r(T_{n_1}),r(T_{n_2}),\cdots$是一个反例序列。但这与 T_{n_1}的定义相矛盾。

注:使用一个较弱的嵌入思想,Kruskal 在*Trans. Amer. Math. Soc.* **95** (1960),210~225 实际上证明了一个更强的结果。尽管结果是模糊地相似的,但他的定理并非直接从无穷性引理推出。确实,König 本人证明了 Kruskal 定理的一个特殊情况,说明没有非负整数的两两不相兼容的 n 元组的无穷序列,这里兼容性指的是一个 n 元组的所有分量都≤其它的相应分量[*Mathematikai és Fizikai Lapok* **39** (1932),27~29]。关于进一步的发展,请见*J. Combinatorial Theory* **A13** (1972),297~305。关于对于算法终止的应用,也请见 N. Dershowitz,*Inf. Proc. Letters* **9** (1979),212~215。

2.3.4.4 小节

1. $\ln A(z)=\ln z+\sum_{k\geq 1}a_k\ln\left(\frac{1}{1-z^k}\right)=\ln z+\sum_{k,t\geq 1}\frac{a_k\ z^{kt}}{t}=\ln z+\sum_{t\geq 1}\frac{A(z^t)}{t}$。

2. 通过微分,以及等置 z^n 的系数,我们得到恒等式

$$na_{n+1}=\sum_{k\geq 1}\sum_{d\backslash k}da_d a_{n+1-k}$$

现在交换求和的次序。

4. a)至少对于 $|z|<1/4$, $A(z)$肯定收敛,因为 a_n 小于有序树 b_{n-1}的个数。由于 $A(1)$是无限的,而且所有系数为正,故有一正数 $\alpha\leq 1$,使得对于 $|z|<\alpha$, $A(z)$收敛,而且在 $z=\alpha$ 处有一奇异点。令 $\psi(z)=A(z)/z$;由于 $\psi(z)>e^{z\psi(z)}$,我们看到 $\psi(z)=m$ 意味着 $z<\ln m/m$,所以 $\psi(z)$是有界的,而且 $\lim_{z\to\alpha-}\psi(z)$存在。于是 $\alpha<1$,而且由阿贝尔极限定理 $a=\alpha\cdot\exp\left(a+\frac{1}{2}A(\alpha^2)+\frac{1}{3}A(\alpha^3)+\cdots\right)$。

b)对于 $|z|<\sqrt{\alpha}$, $A(z^2)$, $A(z^3)$,…是解析的,而且 $(1/2)A(z^2)+(1/3)A(z^3)+\cdots$在一略小的圆盘里一致收敛。

c)如果 $\partial F/\partial\omega=a-1\neq 0$,则隐函数定理意味着,在$(\alpha,a/\alpha)$的一个邻域内,有解析函数 $f(z)$使得 $F(z,f(z))=0$。但这意味着 $f(z)=A(z)/z$,与 $A(z)$在 α 处奇异的事实相矛盾。

d)显然。

e) $\partial F/\partial\omega=A(z)-1$ 且 $|A(z)|<A(\alpha)=1$,因为 $A(z)$的系数都为正。因此如同在 c)中那样,在所有这样的点处,$A(z)$是正则的。

f) 邻近$(\alpha,1/\alpha)$,我们有等式 $0=\beta(z-\alpha)+(\alpha/2)(w-1/\alpha)^2+$ 更高阶的项,其中 $w=(1/z)A(z)$;所以按隐函数定理,w 是 $\sqrt{z-\alpha}$ 的解析函数。因而,有一区域 $|z|<\alpha_1$ 减去一切口$[\alpha,\alpha_1]$,其中 $A(z)$具有所述形式。(由于正号将最终地使系数成为负的,因而选定负号。)

g)所述形式的任何函数,渐近地有系数$\frac{\sqrt{2\beta}}{\alpha^n}\binom{1/2}{n}$。请注意

$$\binom{3/2}{n}=O\left(\frac{1}{n}\binom{1/2}{n}\right)$$

关于进一步的细节,以及自由树之个数的渐近值,见 R. Otter, *Ann. Math.* (2) **49** (1948),583 ~ 599。

5.
$$c_n = \sum_{j_1+2j_2+\cdots=n} \binom{c_1+j_1-1}{j_1}\cdots\binom{c_n+j_n-1}{j_n} - c_n,\quad n>1$$

因此

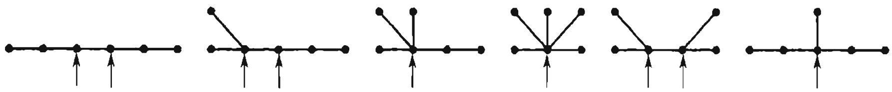
$$2C(z)+1-z = (1-z)^{-c_1}(1-z^2)^{-c_2}(1-z^3)^{-c_3}\cdots = \exp\left(C(z)+\frac{1}{2}C(z^2)+\cdots\right)$$

我们求得 $C(z)=z+z^2+2z^3+5z^4+12z^5+33z^6+90z^7+261z^8+766z^9+\cdots$。当 $n>1$ 时,有 n 个边的串并网络的个数是 $2c_n$[请见 P. A. MacMahon, *Proc. London Math. Soc.* **22** (1891), 330 ~ 339]。

6. $zG(z)^2=2G(z)-z-zG(z^2)$; $G(z)=1+z+z^2+2z^3+3z^4+6z^5+11z^6+23z^7+46z^8+98z^9+\cdots$。函数 $F(z)=1-zG(z)$满足较简单的关系 $F(z^2)=2z+F(z)^2$。[J. H. M. Wedderburn, *Annals of Math.* **24** (1922),121 ~ 140。]

7. $g_n=ca^n n^{-3/2}(1+O(1/n))$,其中 $c\approx 0.7916031835775$, $a\approx 2.483253536173$。

8.



9. 如果有两个形心,则通过考虑从一个形心到另一个的一条通路,我们发现不可能有中间的点,所以任何两个形心是相邻的。一个树包含有三个互相相邻的顶点是不可能的,所以至多有两个。

10. 如果 X, Y 是相邻的,则令 $s(X,Y)$是在 X 的 Y 子树中的顶点数。于是 $s(X,Y)+s(Y,X)=n$。正文中的论证说明,如果 Y 是形心,则 $\text{weight}(X)=s(X,Y)$。因此如果 X 和 Y 两者都是形心,则 $\text{weight}(X)=\text{weight}(Y)=n/2$。

借助于这个记号,进行正文中的论证,也说明如果 $s(X,Y)\geqslant s(Y,X)$,则在 X 的 Y 子树中有一形心。所以如果两个具有 m 个顶点的自由树,通过 X 与 Y 之间的一条边连接,则我们就得到一个自由树,其中 $s(X,Y)=m=s(Y,X)$,而且必然有两个形心(即 X 和 Y)。

[对于所有相邻的 X 和 Y,在 $O(n)$步之内来计算 $s(X,Y)$是一个很好的程序设计的习题;从这个信息出发,我们可以很快地就找到形心。关于形心的位置的一个有效算法,首先是由 A. J. Goldman 给出的,*Transportation Sci.* **5** (1971),212 ~ 221。]

11. $zT(z)^t=T(z)-1$;这样,$z+T(z)^{-t}=T(z)^{1-t}$。由等式 1.2.9-(21),$T(z)=\sum_n A_n(1,-t)z^n$,所以 t 叉树的个数是

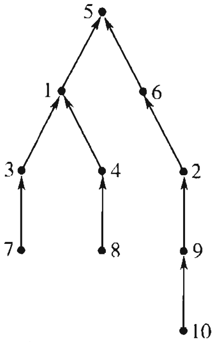
$$\binom{1+tn}{n}\frac{1}{1+tn}=\binom{tn}{n}\frac{1}{(t-1)n+1}$$

12. 考虑有向图,对于所有的 $i\neq j$,它有一条从 V_i 到 V_j 的有向边。习题 2.3.4.2-19 的矩阵 A_0,是组合的$(n-1)\times(n-1)$矩阵,且对角线上是 $n-1$,而 -1 在对角线外。所以其行列式为

$$(n+(n-1)(-1))n^{n-2}=n^{n-2}$$

这是具有一给定的根的有向树的个数。(也可以利用习题 2.3.4.2-20。)

13.



14. 真的,因为根将不变成一片叶,直到撤消所有其它分支为止。

15. 在典型表示 $V_1, V_2, \cdots V_{n-1}$ 中,$f(V_{n-1})$ 是被当做有向图之有向树的一个拓扑排序,但这个顺序一般将不为算法 2.2.3T 所输出。可以改变算法 2.2.3T,使得如果步骤 T6 的"插入队列"的操作为一过程所代替,它确定 V_1, $V_2, \cdots, V_{n-1}$ 的值,这个过程调整链接使得表的条目从前端到后端以递增的次序出现;于是队列就变成优先队列了。

(然而,一般的优先队列不需要求典型表示;我们只需要从 1 到 n 扫描顶点,寻找叶,同时从小于被扫描指针的新叶剪掉通路;见下列习题。)

16. **D1**. 置 C[1]←…←C[n]←0,然后对于 $1 \le j < n$ 置 C[$f(V_j)$]←C[$f(V_j)$]+1。(因此顶点 k 是一个叶当且仅当 C[k]=0。)置 k←0 和 j←1。

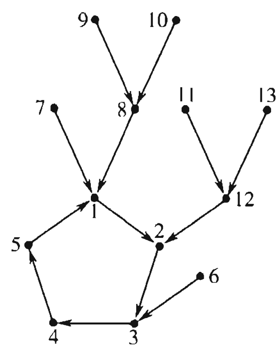
D2. 增加 k 一次或多次直到 C[k]=0 为止,然后置 l←k 。

D3. 置 PARENT[l]←$f(V_j)$,l←$f(V_j)$,C[l]←C[l]−1 以及 j←j+1 。

D4. 如果 $j=n$,则置 PARENT[l]←0 并结束此算法。

D5. 如果 C[l]=0 且 $l<k$,则转到 D3;否则转回 D2。 ▌

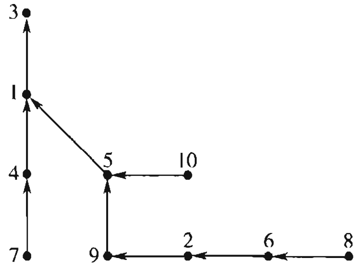
17. 必须恰有一条回路 $x_1, x_2, \cdots, x_k$,其中 $f(x_j) = x_{j+1}$ 和 $f(x_k) = x_1$。我们将枚举所有的 f,它有一条长度 k 的回路,使得每个 x 的迭代最终地进入这条回路。如同正文中那样定义典型表示 $f(V_1), f(V_2), \cdots, f(V_{m-k})$;现在 $f(V_{m-k})$ 在回路中,于是通过写下回路的其余部分 $f(f(V_{m-k})), f(f(f(V_{m-k})))$,等等,我们继续获得一"典型表示"。例如,其图如右图所示的具有 $m=13$ 的函数,导致表示 3,1,8,8,1,12,12,2,3,4,5,1。我们得到一个 $m-1$ 个数的序列,其中最后 k 个是不同的。反之,我们可以从任何这样的序列逆转这个构造(假定 k 已知);因此恰有 $m^{\underline{k}} - m^{m-k-1}$ 个这样的具有一个 k 回路的函数。(相关结果见习题 3.1-14。公式 $m^{m-1}Q(m)$ 首先是由 L. Katz 得到的,*AnnaLs of Math. Statistics* **26** (1955), 512~517。)



18. 为了从序列 $s_1, s_2, \cdots, s_{n-1}$ 重新构造树,以 s_1 作为根开始并逐次地向这树附上那些指向 $s_1, s_2, \cdots$ 的有向边。如果顶点 s_k 事先已经出现,则不命名地保留导向 s_{k-1} 的有向边的初始顶点,否则给这个顶点以名称 s_k。在放置了所有 $n-1$ 条有向边之后,通过使用未曾出现的数来命名还没有名称的所有顶点,并按建立这些无名顶点的顺序,以递增顺序给它们命名。

例如,从 3,1,4,1,5,9,2,6,5 出发我们将构造下页右上所示树。在这个方法与正文中的方法

之间,没有简单的联系。有可能有更多的表示;见 E. H. NevillLe 的论文, *Proc. Cambridge Phil. Soc.* **49** (1953), 381 ~ 385。



19. 典型表示将精确地有 $n-k$ 个不同的值,因此我们可以枚举具有这个性质的 $n-1$ 个数的序列。答案是 $n^{\underline{n-k}}\left\{ {n-1 \atop n-k} \right\}$。

20. 考虑这样树的典型表示。我们是问在$(x_1+\cdots+x_n)^{n-1}$中有多少项具有 k_0 个指数 0, k_1 个指数 1,等等。这显然就是,这种项的系数,乘以这种项的项数,即

$$\frac{(n-1)!}{(0!)^{k_0}(1!)^{k_1}\cdots(n!)^{k_n}}\times\frac{n!}{k_0!\,k_1!\cdots k_n!}$$

21. 具有 $2m$ 个顶点的一个也没有;如果有 $n=2m+1$ 个顶点,则答案可从习题 20 对于 $k_0=m+1, k_2=m$ 得到,即$\binom{2m+1}{m}(2m)!/2^m$。

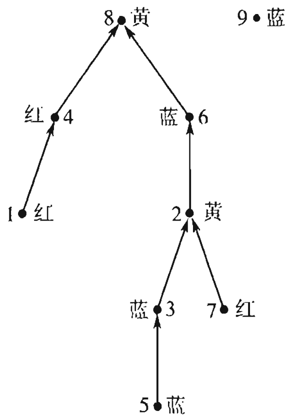
22. 确切地为 n^{n-2};因为如果 X 是一个特殊的顶点,则自由树与以 X 为根的有向树一一对应。

23. 有可能对每一未标号的有序树,以 $n!$ 种方式进行标号,而且这些有标号的有序树,每一个都不相同。所以总数是 $n!\,b_{n-1}=(2n-2)!/(n-1)!$ 。

24. 对于一个给定根,与对于别的根一般多,所以答案一般是 $1/n$ 乘以习题 23 中的答案;而且在此特殊情况下答案为 30。

25. 对于 $0\leqslant q<n$, $r(n,q)=(n-q)n^{q-1}$。(等式(24)当 $s=1$ 时的特殊情况。)

26. $(k=7)$



27. 给定一个从$\{1,2,\cdots,r\}$到$\{1,2,\cdots,q\}$的函数 g,使得从 V_k 到 $U_{g(k)}$加入的一些有向边不引入有向回路,构造一个序列 $a_1,\cdots,a_r$ 如下:如果对于任何 $j\neq k$,没有从 V_j 到 V_k 的有向通路,则称顶点 V_k 是"自由的"。因为没有有向回路,故至少必有一个自由顶点。命 b_1 是使得 V_{b_1}为自由顶点的最小整数;并假定已经选定 $b_1,\cdots,b_t$,命 b_{t+1}是不同于 $b_1,\cdots,b_t$ 的最小整数,对于它,当 $1\leqslant k\leqslant t$ 时,通过删去从 V_{b_k} 到 $U_{g(b_k)}$的有向边所得到的图中,$V_{b_{t+1}}$是自由的。这个规则定义

了整数{1,2,…,r}的一个排列 $b_1b_2\cdots b_r$。对于 $1\leqslant k\leqslant r$,命 $a_k=g(b_k)$;这就定义了一个序列,使得对于 $1\leqslant k<r$,有 $1\leqslant a_k\leqslant q$,而且 $1\leqslant a^r\leqslant p$。

反之,如果给定了这样一个序列 $a_1,\cdots,a_r$,如果不存在这样的 j,使得 $a_j>p$ 和 $f(a_j)=k$,则称顶点 V_k 是“自由的”。由于 $a_r\leqslant p$,至多有 $r-1$ 个非自由的顶点。命 b_1 是使 V_{b_1} 为自由顶点的最小整数;并且假设 $b_1,\cdots,b_t$ 已经选定,命 b_{t+1} 是不同于 $b_1,\cdots,b_t$ 的最小整数,对于它,$V_{b_{t+1}}$ 对于序列 $a_{t+1},\cdots,a_r$ 是自由的。这个规则定义了整数{1,2,…,r}的一个排列 $b_1b_2\cdots b_r$。对于 $1\leqslant k\leqslant r$ 命 $g(b_k)=a_k$;这就定义了一个函数,使得加进从 V_k 到 $U_{g(k)}$ 的一些有向边不引进有向回路。

28. 命 f 是从{2,…,m}到{1,2,…,n}的 n^{m-1} 个函数中的任一个,并考虑具有顶点 $U_1,\cdots,U_m,V_1,\cdots,V_n$ 和对于 $1<k\leqslant m$ 从 U_k 到 $V_{f(k)}$ 的有向边的有向图。以 $p=1,q=m,r=n$ 应用习题 29,以证明有 m^{n-1} 种方法把从诸 V 到诸 U 的更多的有向边加进来,以得到根为 U_1 的有向树。因为在所求的自由树的集合与根为 U_1 的有向树的集合之间,有一一对应关系,故答案是 $n^{m-1}\cdot m^{n-1}$。[这个构造可进行广泛的推广;见 D. E. Knuth, *Canadian J. Math.* **20** (1968), 1077 ~ 1086。]

29. 如果 $y=x^t$,则 $(tz)y=\ln y$,而且我们看到,只须对于 $t=1$ 来证明这恒等式。现在如果 $zx=\ln x$,则我们通过习题 25 知道对于非负整数 m,$x^m=\sum_k E_k(m,1)z^k$。因此

$$x^r=e^{zxr}=\sum_k\frac{(zxr)^k}{k!}=\sum_{j,k}\frac{r^kz^{k+j}E_j(k,1)}{k!}=\sum_k\frac{z^k}{k!}\sum_j\binom{k}{j}j!\,E_j(k-j,1)r^{k-j}=$$

$$\sum_k\frac{z^k}{k!}\sum_j\binom{k-1}{j}k^jr^{k-j}=\sum_k z^kE_k(r,1)$$

[习题 4.7-22 推导出更一般得多的结果。]

30. 所述的每个图定义一个集合 $C_x\subseteq\{1,\cdots,n\}$,这里 j 在 C_x 中,当且仅当,对于某个 $i\leqslant x$,有一条从 t_j 到 r_i 的通路。对于给定的 C_x,所述的每个图由两个独立的部分组成:对于 $i\leqslant x$ 和 $j\in C_x$,关于顶点 r_i,s_{jk},t_j 的 $x(x+\epsilon_1z_1+\cdots+\epsilon_nz_n)^{\epsilon_1+\cdots+\epsilon_n-1}$ 个图中的一个,其中 $\epsilon_j=[j\in C_x]$,加上关于剩下的顶点的 $y(y+(1-\epsilon_1)z_1+\cdots+(1-\epsilon_n)z_n)^{(1-\epsilon_1)+\cdots+(1-\epsilon_n)-1}$ 个图中的一个。

31. $G(z)=z+G(z)^2+G(z)^3+G(z)^4+\cdots=z+G(z)^2/(1-G(z))$。因此 $G(z)=(1/4)(1+z-\sqrt{1-6z+z^2})=z+z^2+3z^3+11z^4+45z^5+\cdots$。[注:等价于这个问题的另一个问题,是 E. Schröder 提出和解决的, *Zeitschrift für Mathematik und Physik* **15** (1870), 361 ~ 376,他确定出在凸$(n+1)$边形中插入非重叠的对角线的方法种数。对于 $n>1$,这些数恰是习题 2.2.1-11 中所得的值之一半,因为 Pratt 文法允许相关联的分析树的根节点有度数 1。习题 2.2.1-12 中计算了这个近似值。奇怪的是值 $[z^{10}]G(z)=103049$ 似乎在公元前二世纪就已被 Hipparchus,作为“只从 10 个简单的命题可以构造的肯定的复合命题”的个数计算过了。请见 R. P. Stanley, *AMM.* **104** (1997), 344 ~ 350。]

32. 如果 $n_0\neq1+n_2+2n_3+3n_4+\cdots$,则为 0(参照习题 2.3-21),否则为 $(n_0+n_1+\cdots+n_m-1)!/n_0!\,n_1!\cdots n_m!$。

为了证明这一结果,我们回想起,一个具有 $n=n_0+n_1+\cdots+n_m$ 个节点的未标号的树,通过在后根序下节点的度数的序列 $d_1d_2\cdots d_n$ 来表征之(2.3.3 小节)。而且,这样一个度数的序列,对应于一个树,当且仅当,对于 $0<k\leqslant n$,$\sum_{j=1}^{k}(1-d_j)>0$。(波兰后缀记号的这一重要性质,通过归纳法很容易证明;参见带有建立一个树的函数 f(就像在 2.3.2 小节中的 TREE 函数那样)的算法 2.3.3F。)特别是,d_1 必须为 0。因此我们问题的答案是,对于 $j>0$,具有 j 的 n_j 次出现的序列

$d_2 \cdots d_n$ 的个数,即多项式系数

$$\binom{n-1}{n_0-1, n_1, \cdots, n_m}$$

减去这样的序列 $d_2 \cdots d_n$ 的个数,对于这些序列,当某个 $k \geq 2$ 时,有 $\sum_{j=2}^{k}(1-d_j) < 0$。

我们可以枚举后边的序列如下:命 t 是使得 $\sum_{j=2}^{t}(1-d_j) < 0$ 之极小者;则 $\sum_{j=2}^{t}(1-d_j) = -s$,其中 $1 \leq s < d_t$,而且我们可以形成子序列 $d_2' \cdots d_n' = d_{t-1} \cdots d_2 0 d_{t+1} \cdots d_n$,对于 $j \neq d_t$,它有 j 的 n_j 个出现,对于 $j = d_t$,它有 j 的 $n_j - 1$ 个出现。现在 $\sum_{j=2}^{k}(1-d_j')$,当 $k = n$ 时,等于 d_t,当 $k = t$ 时,等于 $d_t - s$;当 $k < t$ 时,它是

$$\sum_{2 \leq j < t}(1-d_j) - \sum_{2 \leq j \leq t-k}(1-d_j) \leq \sum_{2 \leq j < t}(1-d_j) = d_t - s - 1$$

由此得出,给定 s 和任何序列 $d_2' \cdots d_n'$,这个构造可加以逆转;因此有一个给定的 d_t 和 s 之值的序列 $d_2 \cdots d_n$ 的个数,是多项式系数

$$\binom{n-1}{n_0, \cdots, n_{d_t}-1, \cdots, n_m}$$

因此,通过对所有可能的 d_t 和 s 之值进行求和,就得到了对应于这些树的序列 $d_2 \cdots d_n$ 的个数:

$$\sum_{j=0}^{m}(1-j)\binom{n-1}{n_0, \cdots, n_j-1, \cdots, n_m} = \frac{(n-1)!}{n_0!\, n_1! \cdots n_m!}\sum_{j=0}^{m}(1-j)n_j$$

而且后面的和为 1。

这个结果的一个甚至更为简单的证明,已经由 G. N. Raney 给出(*Transactions of the American Math. Society* **94** (1960),441 ~ 451)。如果 $d_1 d_2 \cdots d_n$ 是具有 j 的 n_j 个出现的任何序列,则恰有对应于一树的循环的重新配置 $d_k \cdots d_n d_1 \cdots d_{k-1}$,也就是其中 k 是使得 $\sum_{j=1}^{k-1}(1-d_j)$ 为极小时的极大值的重新配置。[在二叉树情况下的这一论证看来首先是由 C. S. Peirce 在他未发表的手稿中发现的;请见他的*New ELements of Mathematics* **4** (The Hague: Mouton, 1976),303 ~ 304。在 t 叉树的情况是由 Dvoretzky 和 Motzkin 发现的,*Duke. Math. J.* **14** (1947),305 ~ 313。]

还有另一个由 G. Bergman 给出的证明,如果 $d_k > 0$,则以 $(d_k + d_{k+1} - 1)$ 归纳地代替 $d_k d_{k+1}$ [*Algebra Universalis* **8** (1978),129 ~ 130]。

以上这些方法都可加以推广来证明,假定条件 $n_0 = f + n_2 + 2n_3 + \cdots$ 满足,有 f 个树和度数 j 的 n_j 个节点的(有序,无标号的)森林的个数是 $(n-1)!\,f/n_0!\, n_1! \cdots n_m!$。

33. 考虑具有标号为 1 的 n_1 个节点,标号为 2 的 n_2 个节点,…,且使标号为 j 的每个节点有度数 e_j 的树的个数。命这个数为 $c(n_1, n_2, \cdots)$,而且所说明的度数 $e_1, e_2, \cdots$ 被看成是固定的。生成函数 $G(z_1, z_2, \cdots) = \sum c(n_1, n_2, \cdots) z_1^{n_1} z_2^{n_2} \cdots$ 满足恒等式 $G = z_1 G^{e_1} + \cdots + z_r G^{e_r}$,因为 $z_j G^{e_j}$ 枚举其根标号为 j 的那些树。而且由前面习题的结果,

$$c(n_1, n_2, \cdots) = \begin{cases} \dfrac{(n_1 + n_2 + \cdots - 1)!}{n_1!\, n_2! \cdots}, & \text{如果}(1-e_1)n_1 + (1-e_2)n_2 + \cdots = 1 \\ 0, & \text{否则} \end{cases}$$

更一般地,因为 G^f 枚举有这样标号的有序森林的个数,故对于整数 $f > 0$ 我们有

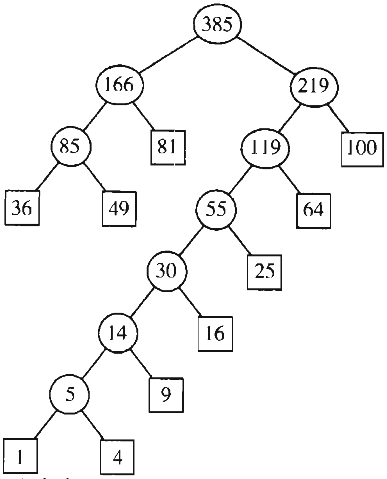
$$w^f = \sum_{f = (1-e_1)n_1 + (1-e_2)n_2 + \cdots} \frac{(n_1 + n_2 + \cdots - 1)!\, f}{n_1!\, n_2! \cdots} z_1^{n_1} z_2^{n_2} \cdots$$

当 $r = \infty$ 时,这些公式是有意义的,而且实际上它们等价于拉格朗日的反演公式。

2.3.4.5 小节

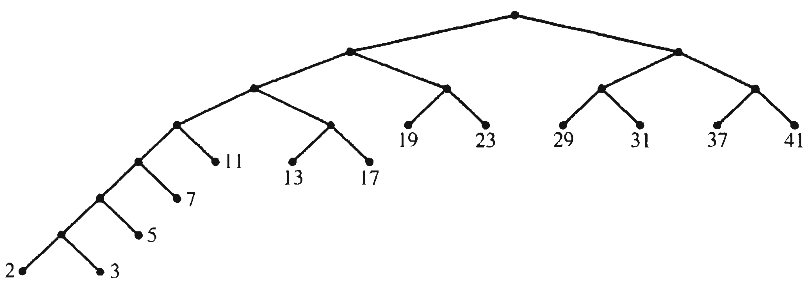
1. 全部有$\binom{8}{5}$,因为编号为 8,9,10,11,12 的节点,可以附加于 4,5,6 和 7 之下的 8 个位置中的任一个。

2.



3. 通过对 m 用归纳法,这个条件是必要的。反之,如果$\sum_{j=1}^{m}2^{-l_j}=1$,则我们要构造具有通路长度 $l_1,\cdots,l_m$ 的扩充的二叉树。当 $m=1$ 时,我们有 $l_1=1$ 而且这构造是平凡的。否则我们可以假定诸 l 是有序的,使得对于 $1\leqslant q\leqslant m$ 的某 q, $l_1=l_2=\cdots=l_q>l_{q+1}\geqslant l_{q+2}\geqslant\cdots\geqslant l_m>0$。现在 $2^{l_1-1}=\sum_{j=1}^{m}2^{l_1-l_j-1}=\frac{1}{2}q+$整数,因此 q 为偶数。通过对 m 用归纳法,有一通路长度为 $l_1-1,l_3,l_4,\cdots,l_m$ 的树;取这样一个树并且以其儿子在 $l_1=l_2$ 层上的一个内部节点来代替在 l_1-1层的外部节点之一。

4. 首先,通过哈夫曼方法找一个树。如果 $w_j<w_{j+1}$,则 $l_j\geqslant l_{j+1}$,因为这个树是最优的。习题 3 答案中的构造,现在为我们提供了有这些相同通路长度而且具有在适当顺序下的权数的另一棵树。例如,树(11)变成



5. a) $b_{np}=\sum_{\substack{k+l=n-1\\ r+s+n-1=p}}b_{kr}b_{ls}$。因此 $zB(w,wz)^2=B(w,z)-1$。

b)关于 w 取偏导数:

$$2zB(w,wz)(B_w(w,wz)+zB_z(w,wz))=B_w(w,z)$$

因此如果 $H(z)=B_w(1,z)=\sum_n h_n z^n$，则我们求得 $H(z)=2zB(z)(H(z)+zB'(z))$；而且对于 $B(z)$的已知的公式导出

$$H(z)=\frac{1}{1-4z}-\frac{1}{z}\left(\frac{1-z}{\sqrt{1-4z}}-1\right),\quad \text{所以}\quad h_n=4^n-\frac{3n+1}{n+1}\binom{2n}{n}$$

平均值是 h_n/b_n。c)近似地，这得出 $n\sqrt{\pi n}-3n+O(\sqrt{n})$。

关于一些类似问题的解，见 John Riordan, *IBM J. Res. and Devel.* **4** (1960), 473～478；A. Rényi 和 G. Szekeres, *J. Australian Math. Soc.* **7** (1967), 497～507；John Riordan 和 N. J. A. Sloane, *J. Australian Math. Soc.* **10** (1969), 278～282；以及习题 2.3.1-11。

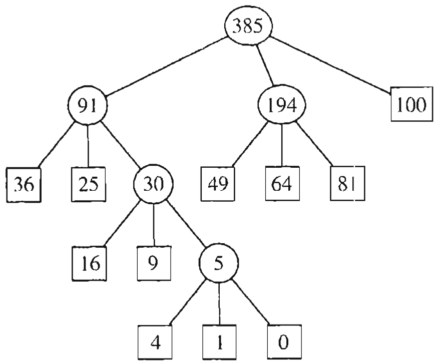
6. $n+s-1=tn$。

7. $E=(t-1)I+tn$。

8. 通过分部求和给出$\sum_{k=1}^{n}\lfloor\log_t((t-1)k)\rfloor=nq-\sum k$，其中右边的求和是对于使得 $0\le k\le n$ 的 k 值和对于某个 j，$(t-1)k+1=t^j$ 进行的。后一个和可重新写成$\sum_{j=1}^{q}(t^j-1)/(t-1)$。

9. 对树大小用归纳法。

10. 必要时，通过加上额外的零权数，我们可以假定 $m \bmod (t-1)=1$。为得到具有极小的加权通路长度的 t 叉树，在每一步骤都组合最小的 t 个值，并以它们的和来代替它们。证明实质上与二叉树相同。所求的三叉树为右下图所示。



黄光明已经发现[*SIAM J. Appl. Math.* **37** (1979), 124～127]，对于有任何规定的度数的多重集合的极小带权通路长度树而言，类似的过程正确：在每步骤中，组合最小的 t 个权，其中 t 要尽可能地小。

11. "杜威记号"是节点号数的二进制表示。

12. 由习题 9，它是内部通路长度除以 n，加上 1。(这个结果对于一般树以及二叉树都成立。)

13. [请见 J. Van Leeuwen, *Proc. 3rd International Colloq. Automata, Languages and Programming* (Edinburgh University Press, 1976), 382～410。]

H1. [初始化] 对于 $1\le i\le m$ 置 $A[m-1+i]\leftarrow w_i$；然后置 $A[2m]\leftarrow\infty$，$x\leftarrow m$，$i\leftarrow m+1$，$j\leftarrow m-1$，$k\leftarrow m$。(在此算法期间 $A[i]\le\cdots\le A[2m-1]$是未被使用的外部权的队列；$A[k]\ge\cdots\ge A[j]$是未被使用的内部权的队列，如果 $j<k$ 则为空；当前左和右指针为 x 和 y。)

H2. [寻找右指针] 如果 $j<k$ 或$A[i]\le A[j]$，则置 $y\leftarrow i$ 和 $i\leftarrow i+1$；否则置 $y\leftarrow j$ 和 $j\leftarrow j-1$。

H3. [建立内部节点] 置 $k\leftarrow k-1$，$L[k]\leftarrow x$，$R[k]\leftarrow y$，$A[k]\leftarrow A[x]+A[y]$。

H4. [完成了吗?] 如果 $k=1$ 则终止算法。

H5. [寻找左指针] (这时 $j\ge k$ 而且队列包含总数为 k 的未被使用的权。如果 $A[y]<0$ 我们有 $j=k$，$i=y+1$ 而且 $A[i]>A[j]$。)如果 $A[i]\le A[j]$，则置 $x\leftarrow i$ 和 $i\leftarrow i+1$；否则置 $x\leftarrow j$ 和 $j\leftarrow j-1$，返回步骤 H2。▌

14. 只须作少量修改，这个证明对于 $k=m-1$ 适用。[请见*SIAM J. Appl. Math.* **21** (1971), 518。]

15. 分别使用组合的权函数(a)$1+\max(w_1,w_2)$和(b)xw_1+xw_2代替(9)中的 w_1+w_2。[(a)部分是由 M. C. Golumbic 给出的,*IEEE Trans*. **C-25** (1976), 1164~1167,(b)部分是由 T. C. Hu,D. Kleitman 和 J. K. Tamaki 给出的,*SIAM. J. Appl. Math*. **37** (1979),246~256。哈夫曼问题是当 $x\to 1$ 时(b)的极限情况,因为 $\sum(1+\epsilon)^{l_j}w_j=\sum w_j+\epsilon\sum w_jl_j+O(\epsilon^2)$。]

D. Stott Parker,Jr. 已经指出,如果在(b)部分的每一步骤中,组合两个极大的权,当 $0<x<1$ 时,类似哈夫曼的算法也将找出 $w_1x^{l_1}+\cdots+w_mx^{l_m}$的极小值。特别是,当 $w_1\leqslant\cdots\leqslant w_m$ 时,$w_12^{-l_1}+\cdots+w_m2^{-l_m}$的极小值是 $w_1/2+\cdots+w_{m-1}/2^{m-1}+w_m/2^{m-1}$。关于进一步的推广,也见 D. E. Knuth,*J. Comb. Theory* **A32** (1982),216~224。

16. 设 $l_{m+1}=l'_{m+1}=0$,则

$$\sum_{j=1}^{m}w_jl_j\leqslant\sum_{j=1}^{m}w_jl'_j=\sum_{k=1}^{m}(l'_k-l'_{k+1})\sum_{j=1}^{k}w_j\leqslant\sum_{k=1}^{m}(l'_k-l'_{k+1})\sum_{j=1}^{k}w'_j=\sum_{j=1}^{m}w'_jl'_j$$

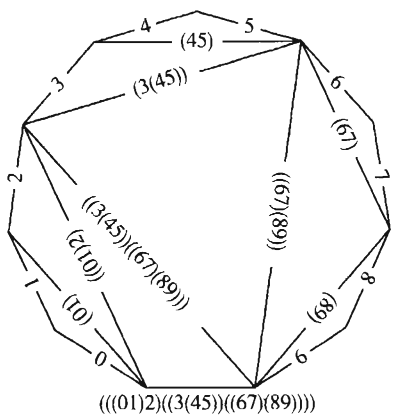
因为如同在习题 4 那样,$l'_j\geqslant l'_{j+1}$。对于许多其它类型的最优树,包括习题 10 的那些在内,同一证明成立。

17. (a)这是习题 14。(b)我们可以把 $f(n)$推广到凹函数 $f(x)$,所以所述不等式成立。现在 $F(m)$是$\sum_{j=1}^{m-1}f(s_j)$的极小值,其中 s_j 是在权 1,1,…,1 上的扩充二叉树的内部节点的权。哈夫曼算法在这种情况下是构造有 $m-1$ 个内部节点的完全二叉树的,它产生出最优树来。$k=2^{\lceil\lg(n/3)\rceil}$的选择定义具有相同的内部权的二叉树,所以对于每个 n,它在递推中产生极小值。[*SIAM J. Appl. Math*. **31** (1976),368~378。]我们可以在 $O(\log n)$步内计算 $F(n)$的值;见习题 5.2.3-20 和习题 5.2.3-21。如果 $f(n)$是凸的而不是凹的,使得 $\Delta^2f(n)\geqslant 0$,则当 $k=\lfloor n/2\rfloor$时得到递推的解。

2.3.4.6. 小节

1. 选择多边形的一个边并把它叫做底。给定一个三角划分,令在底上的三角形对应于一个二叉树的根,并令该三角形的其它两边定义左和右子多边形的底,它们以同样的方式对应于左和右子树。我们递归地进行直到达到"二边"的多边形为止,它对应于空的二叉树。

以另外一种方式来表示这种对应性,我们可以以整数 0,…,n 来对三角化了的多边形的非底的边进行标号;而且当三角形的两个相邻的边按顺时针顺序标号为 α 和β 时,我们可以把第三边标号为($\alpha\beta$),底的标号于是表征了二叉树和三角划分。例如,



对应于 2.3.1-(1)中的二叉树。

2.a) 像在习题 1 中那样取一个底边，如果该边是在分割的 r 边形中的 $(d+1)$ 边形的一部分，则给它 d 个后裔。于是其它的 d 个边是子树的底。这就定义了在 Kirkman 问题和有 $r-1$ 个叶及 $k+1$ 个非叶，且无度数为 1 的节点的所有有序树之间的一种对应。(当 $k=r-3$ 时，我们有习题 1 的情况。)

b) 有 $\binom{r+k}{k+1}\binom{r-3}{k}$ 个非负整数序列 $d_1\,d_2\cdots d_{r+k}$ 使得诸 d 当中有 $r-1$ 个为 0，其中无一个为 1，而和数为 $r+k-1$。精确地只有一个循环排列 $d_1 d_2\cdots d_{r+k}$，$d_2\cdots d_{r+k}\,d_1$，$\cdots$，$d_{r+k}\,d_1\cdots d_{r+k-1}$ 满足对于 $1\leqslant q\leqslant r+k$，$\sum_{j=1}^{q}(1-d_j)>0$ 这一附加的性质。

[Kirkman 在 *Philos. Trans.* **147** (1857), 217～272, §22 中给出了对于他的猜测的证据。Cayley 在 *Proc. London Math. Soc.* **22** (1891), 237～262 中证明了它但没有注意它同树的联系。]

3. a) 令诸顶点是 $\{1,2,\cdots,n\}$。如果 i 和 j 是同一部分的相邻元素且 $i<j$，就画从 i 到 j 的一个 RLINK。如果 $j+1$ 是该部分的最小数，就从 j 到 $j+1$ 画一个 LLINK。于是有 $k-1$ 个非空的 LLINK，$n-k$ 个非空的 RLINK，而且我们有在先根序下其节点为 $1\ 2\ \cdots\ n$ 的二叉树。利用 2.3.2 小节的自然对应，这个规则定义了在"n 边形顶点分划成 k 个不相交部分的分划"与"有 n 个顶点和 $n-k+1$ 个叶的森林"之间的一一对应。交换 LLINK 和 RLINK 也给出"有 n 个顶点和 k 个叶"的森林。

b) 有 n 个顶点和 k 个叶的森林也对应于嵌套圆括弧的一个序列，它含 n 个左圆括弧，n 个右圆括弧和 k 个"()"的出现。我们能够枚举这样的序列如下：

我们说 0 和 1 的串是 (m,n,k) 的串，如果有 m 个 0，n 个 1 和 k 个"01"的出现。于是 0010101001110 是 (7,6,4) 串。(m,n,k) 串的个数为 $\binom{m}{k}\binom{n}{k}$，因为我们可自由地选择哪一个 0 和 1 将形成 01 的对。

令 $S(\alpha)$ 是在 α 中 0 的个数减去 1 的个数。我们说一个串 σ 是好的，如果 α 是 σ 的一个前缀时 $S(\alpha)\geqslant 0$（换句话说，如果 $\sigma=\alpha\beta$ 意味着 $S(\alpha)\geqslant 0$）；否则就说 σ 是坏的。下列对习题 2.2.1-4 的"反射原理"的变形确定坏的 (n,n,k) 串与任意的 $(n-1,n+1,k)$ 串之间的一一对应。

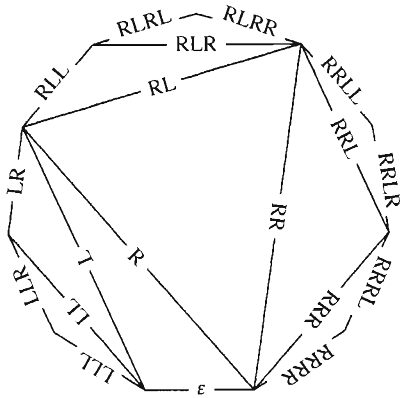
任何坏的 (n,n,k) 串 σ 可惟一地写成 $\sigma=\alpha 0\beta$ 的形式，其中 $\bar{\alpha}^R$ 和 β 都是好的。(这里 $\bar{\alpha}^R$ 是由 α 通过把它颠倒过来并且对其所有二进位取补所得到的串。)于是 $\sigma'=\alpha 1\beta$ 是 $(n-1,n+1,k)$ 串。反之，每一个 $(n-1,n+1,k)$ 串可以惟一地写成 $\alpha 1\beta$ 的形式，其中 $\bar{\alpha}^R$ 和 β 是好的，而 $\alpha 0\beta$ 则是一个坏的 (n,n,k) 串。

所以，具有 n 个顶点和 k 个叶的森林的数目是 $\binom{n}{k}\binom{n}{k}-\binom{n-1}{k}\binom{n+1}{k}=n!\,(n-1)!/(n-k+1)!\,(n-k)!\,k!\,(k-1)!$。

注：G. Kreweras, *Discrete Math.* **1** (1972), 333～350 以不同的方法枚举了不相交的分划。通过修改分划的偏序导致了森林的一个有趣的偏序，它不同于在习题 2.3.3-19 中的讨论；请见 Y. Poupard, *Cahiers du Bureau Univ. de Recherche Opérationnelle* **16** (1971)，第 8 章；*Discrete Math.* **2** (1972), 279～288；P. Edelman, *Discrete Math.* **31** (1980), 171～180, **40** (1982), 171～179。

定义森林的自然格顺序的第三个方法是由 R. Stanley 在 *Fibonacci Quarterly* **13** (1975), 215～232 引入的。假设我们像上面那样通过以 0 和 1 表示左和右圆括弧的串 σ 来表示森林；于是 $\sigma\leqslant\sigma'$ 当且仅当对于所有的 k，$S(\sigma_k)\leqslant S(\sigma'_k)$，其中 σ_k 表示 σ 的前 k 个二进位。不像其它两个那样，Stanley 的格是分布的。

4. 设 $m=n+2$;由习题1,我们要有三角划分的 m 边形和 $m-1$ 行的中楣之间的一个对应。首先,通过给出对三角划分的边的"由顶向下"的标号而不是早先所考虑的"由底向上"的标号,让我们更仔细地考虑以前的对应:对底赋以空的标号 ϵ,而后递归地对其底为标号 α 的一个三角形的相对立的边标号 αL 和 αR。例如,在这些新约定下前边的框图变成



如果在这个例子中的底边称为10,而其它边如同以前一样为0到9,我们就可以写 $0=10LLL$,$1=10LLR$,$2=10LR$,$3=10RLL$,等等。任何其它的边也可选作底;于是,如果选定0,我们就有 $1=0L$,$2=0RL$,$3=0RRLLL$,等等。不难验证,如果 $u=v\alpha$,我们有 $v=u\alpha^{\mathrm{T}}$,其中 α^{T} 是通过从右到左来读 α 而且把 L 和 R 交换得来。例如 $10=0RRR=1LRR=2LR=3RRL$,等等。如果 u,v 和 w 是多边形的边且 $w=u\alpha L\gamma$ 和 $w=v\beta R\gamma$,则 $u=v\beta L\alpha^{\mathrm{T}}$ 和 $v=u\alpha R\beta^{\mathrm{T}}$。

给定其边被编号为 $0,1,\cdots,m-1$ 的多边形的三角划分,对于任何不相同的边 u 和 v 的任何对,我们定义 (u,v) 如下:令 $u=v\alpha$,并把 α 解释为一个 2×2 矩阵,通过令 $L=\begin{pmatrix}1&1\\0&1\end{pmatrix}$ 和 $R=\begin{pmatrix}1&0\\1&1\end{pmatrix}$ 来进行。于是 (u,v) 被定义为 α 的左上角的元素。注意 α^{T} 是矩阵 α 的转置,因为 $R=L^{\mathrm{T}}$;因此我们有 $(v,u)=(u,v)$。还注意 $(u,v)=1$,当且仅当 u_- 和 v_- 是通过三角划分的一个边连接的,其中 u_- 表示边 u 和 $u-1$ 之间的顶点。

对于所有多边形的边 u 令 $(u,v)=0$。我们现在可以证明 $v=u\alpha$ 意味着

$$\alpha=\begin{pmatrix}(u,v)&(u,v+1)\\(u+1,v)&(u+1,v+1)\end{pmatrix}\quad 对于所有\ u\neq v \qquad (*)$$

其中 $u+1$ 和 $v+1$ 是 u 和 v 的顺时针的后继。证明是通过对 m 的归纳法进行:当 $m=2$ 时等式(∗)是平凡的,因为两条平行的边通过 $u=v\epsilon$ 来相互关联且 $\alpha=\epsilon$ 是单位矩阵。如果任何三角划分通过以一个三角形 $v\,v'v''$ 扩充某边 v 而增大,则 $v=u\alpha$ 意味着 $v'=u\alpha L$ 和 $v''=u\alpha R$;因此 (u,v') 和 (u,v'') 在扩充的多边形中分别等于在原来的三角划分中的 (u,v) 和 $(u,v)+(u,v+1)$。由此得出

$$\alpha L=\begin{pmatrix}(u,v')&(u,v'')\\(u+1,v')&(u+1,v'')\end{pmatrix} 和\ \alpha R=\begin{pmatrix}(u,v'')&(u,v''+1)\\(u+1,v'')&(u+1,v''+1)\end{pmatrix}$$

因此(∗)在扩充的多边形中保持为真。

对应于给定的三角划分的中楣模式现在定义为周期序列

$(0,1)\quad(1,2)\quad(2,3)\quad\cdots\quad(m-1,0)\quad(0,1)\quad(1,2)\quad\cdots$

$(0,2)\quad(1,3)\quad(2,4)\quad\cdots\quad(m-1,1)\quad(0,2)\quad(1,3)\quad\cdots$

$(m-1,2)\quad(0,3)\quad(1,4)\quad\cdots\quad(m-2,1)\quad(m-1,2)\quad(0,3)\quad\cdots$

$(m-1,3)\quad(0,4)\quad(1,5)\quad\cdots\quad(m-2,2)\quad(m-1,3)\quad(0,4)\quad\cdots$

等等直到 $m-1$ 行全都定义了为止。当 $m>3$ 时最后一行以 $(\lceil m/2\rceil+1,\lceil m/2\rceil)$ 开始。条件(*)证明,这个模式是一个中楣,即

$$(u,v)(u+1,v+1)-(u,v+1)(u+1,v)=1 \qquad (**)$$

因为 $\det L=\det R=1$ 意味着 $\det \alpha=1$。我们的例子的三角划分产生

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ···

1 2 4 2 1 5 1 3 1 4 3 1 2 4 2 1 5 1 3 1 4 ···

2 1 7 7 1 4 4 2 2 3 11 2 1 7 7 1 4 4 2 2 3 ···

1 3 12 3 3 3 7 1 5 8 7 1 3 12 3 3 3 7 1 5 8 ···

3 2 5 5 8 2 5 3 2 13 5 3 2 5 5 8 2 5 3 2 13 ···

5 3 2 13 5 3 2 5 5 8 2 5 3 2 13 5 3 2 5 5 8 ···

3 7 1 5 8 7 1 3 12 3 3 3 7 1 5 8 7 1 3 12 3 ···

4 2 2 3 11 2 1 7 7 1 4 4 2 2 3 11 2 1 7 7 1 ···

5 1 3 1 4 3 1 2 4 2 1 5 1 3 1 4 3 1 2 4 2 ···

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ···

关系 $(u,v)=1$ 定义三角划分的边,因此不同的三角划分产生不同的中楣。为了完成一对一对应的证明,我们必须证明正整数的每个 $(m-1)$ 行的中楣模式是由某个三角划分以这个方法得到的。

给定 $m-1$ 行的任何中楣,通过在顶部放置一个新行 0 和在底部放置第 m 行来扩充它,两行全由 0 组成。现在令 0 行的元素称为 $(0,0),(1,1),(2,2)$ 等等。而且对于所有非负整数 $u<v\leqslant u+m$,令 (u,v) 是在 (u,u) 的东南的对角中的元素和在 (v,v) 的西南对角的元素。由假定,对于所有 $u<v<u+m$ 条件(**)成立,事实上我们可以把(**)推广成更一般得多的关系:

$$(t,u)(v,w)+(t,w)(u,v)=(t,v)(u,w)\ \text{对于}\ t\leqslant u\leqslant v\leqslant w\leqslant t+m \qquad (***)$$

因为如果(***)为假,令 (t,u,v,w) 是 $(w-t)m+u-t+w-v$ 有最小值的一个反例。第一种情况:$t+1<u$。于是对于 $(t,t+1,v,w)$,$(t,t+1,u,v)$ 和 $(t+1,u,v,w)$,(***)成立,所以我们求得 $((t,u)(v,w)+(t,w)(u,v))(t+1,v)=(t,v)(u,w)(t+1,v)$;这意味着 $(t+1,v)=0$,矛盾。第二种情况:$v+1<w$。于是对于 $(t,u,w-1,w)$,$(u,v,w-1,w)$ 以及 $(t,u,v,w-1)$,(***)成立;我们得到一个类似的矛盾 $(u,w-1)=0$。第三种情况:$u=t+1$ 和 $w=v+1$。在此情况下,(***)归结为(**)。

现在在(***)中我们置 $u=t+1$ 和 $w=t+m$,并对于 $t\leqslant v\leqslant t+m$ 得到 $(t,v)=(v,t+m)$,因为 $(t+1,t+m)=1$ 且 $(t,t+m)=0$。我们由此结论,任何 $(m-1)$ 行的中楣的条目是周期的:$(u,v)=(v,u+m)=(u+m,v+m)=(v+m,u+2m)=\cdots$。

正整数的每个中楣模式都在第 2 行包含一个 1。因为在(***)中如果我们置 $t=0,v=u+1$ 和 $w=u+2$,我们得到 $(0,u+1)(u,u+2)=(0,u)+(0,u+2)$,因此 $(0,u+2)-(0,u+1)\geqslant(0,u+1)-(0,u)$ 当且仅当 $(u,u+2)\geqslant 2$。这在范围 $0\leqslant u\leqslant m-2$ 中不能对所有 u 成立,因为 $(0,1)-(0,0)=1$ 且 $(0,m)-(0,m-1)=-1$。

最后,如果 $m>3$,在第 2 行中我们不能有连续的 1,因为 $(u,u+2)=(u+1,u+3)=1$ 意味

着$(u,u+3)=0$。因此我们可以把这个中楣归结成为 m 减去 1 的另一个中楣,如同这里所图示的 7 行减为 6 行:

```
1   1   1   1   1   1   1   1   ...
  a   b   c  d+1  1  e+1  y   z   ...
    p   q  c+r  d   e  u+y  v   w   ...
      u  q+v  r   s   u  q+v  r   s   ...
       u+y  v   w   p   q  c+r  d   e   ...
          y   z   a   b   c  d+1  1  e+1 ...
            1   1   1   1   1   1   1   1   ...
```

```
1   1   1   1   1   1   1   ...
  a   b   c   d   e   y   z   ...
    p   q   r   s   u   v   w   ...
      u   v   w   p   q   r   s   ...
        y   z   a   b   c   d   e   ...
          1   1   1   1   1   1   1   ...
```

由归纳法,归约了的中楣对应于一个三角划分,而未归约的中楣对应于多附加一个三角形。[*Math. Gazette* **57** (1974),87 ~ 94,175 ~ 183;Conway 和 Guy,*The Book of Numbers* (New York:Copernicus, 1996), 74 ~ 76,96 ~ 97,101 ~ 102。]

注:这个证明显示了每当(t,u,v,w)是在顺时针顺序下的多边形的边时,通过 2×2 矩阵定义的在任何三角划分上的函数(u,v)满足(***)。我们可以把每个(u,v)表达作为数 $a_j=(j-1,j+1)$中的多项式;这些多项式实际上等同于在 4.5.3 小节中讨论的"连续音",但个别项的符号除外。事实上,$(j,k)=i^{1-k+j}K_{k-j-1}(ia_{j+1},ia_{j+2},\cdots,ia_{k-1})$。因此(***)等价于习题 4.5.3-32 的答案中对于连续音的欧拉恒等式。矩阵 L 和 R 有以下有趣的性质,即行列式值为 1 的任何 2×2 的非负整数的矩阵都可惟一地表示为 L 和 R 的乘积。

还存在许多其它有趣的关系,例如,在一个整数中楣的第 2 行中的数累计接触相应的三角划分多边形每个顶点的三角形个数。在基本区域 $0\leqslant u<v-1<m-1$ 和$(u,v)\neq(0,\ m-1)$中$(u,v)=1$的出现的总数是三角划分的对角线(弦)的个数,即 $m-3=n-1$。2 的总数也是 $n-1$,因为$(u,v)=2$,当且仅当 u_- 和 v_- 是相邻于一个弦的两个三角形的对立顶点。

对(u,v)的另一个解释是由 D. Broline, D. W. Crowe 和 I. M. Jsaacs 发现的[*Geometriæ Dedicata* **3** (1974),171 ~ 176]:它是对于相邻于 $v-u-1$ 个顶点的不同三角形在边 u 和 $v-1$ 之间匹配这些顶点的方法数。

2.3.5 小节

1. 列表结构是这样的有向图,其中离开每个顶点的有向边是有序的,并且有出度 0 的某些顶点被标记为"原子"。而且,有一个顶点 S,使得对于所有顶点 $V\neq S$,有一条从 S 到 V 的有向通路。(对于逆于有向边的方向,S 将是一个"根"。)

2. 不是在相同的方式下,因为在通常表示下的穿线链接,回过头来引向对于子列表来说不是惟一的"PARENT"。习题 2.3.4.2-25 所讨论的表示,或者某个类似的方法,也许可用(但在写作本书的时候,这一方法尚未开发出来)。

3. 如同在正文中提到的那样,我们也证明在结束时,P = P0。如果仅有 P0 有待加标记,则这算法肯定能正确地工作。如果有 $n>1$ 个节点有待标记,我们必须有 ATOM(P0) = 0。然后步骤 E4 置 ALINK(P0)←Λ 和以 ALINK(P0)代替 P0,以及以 P0 代替 T 来执行算法。由归纳法(注意由于 MARK(P0)现在为 1,故由步骤 E4 和 E5,所有对于 P0 的链接都等价于 Λ),我们看到,最终地我们将标记以 ALINK(P0)开始和不通过 P0 之诸通路上的所有节点;然后我们将以 T = P0 和 P = ALINK(P0)达到步骤 E6。现在由于 ATOM(T) = 1,故步骤 E6 恢复 ALINK(P0)和 ATOM(P0),而且我们到达步骤 E5。步骤 E5 置 BLINK(P0)←Λ,等等,而且一个类似的论证表明,我们将最终地标记以 BLINK(P0)开始且不通过 P0 或者从 ALINK(P0)可以到达的节点之诸通路上的所有节点。然后我们将以 T = P0,P = BLINK(P0)到达步骤 E6,而且最后我们以 T = Λ,P = P0 到达步骤

E6。

4. 下列程序加上了在处理原子速度方面所提议的改进，这一改进是在正文中，在算法 E 的陈述之后提出的。

在算法的步骤 E4 和 E5 中，我们要测试是否 MARK(Q) = 0。如果 NODE(Q) = +0，则这是一种非常情况，通过置它等于 −0 而且就像它开始时就是 −0 那样来看待，就可以适当地加以处理，因为它有 ALINK 和 BLINK 两者均为 Λ。这一简化在下面的计时计算中未反映出来。

rI1≡P，rI2≡T，rI3≡Q，rX≡−1(为设置诸 MARK)。

```
      MARK   EQU   0:0
      ATOM   EQU   1:1
      ALINK  EQU   2:3
      BLINK  EQU   4:5
E1    LD1    P0              1          E1. 初始化。P←P0
      ENT2   0               1          T←Λ
      ENTX   -1              1          rX←-1
E2    STX    0,1(MARK)       1          E2. 标记。MARK(P)←1
E3    LDA    0,1(ATOM)       1          E3. 是原子吗?
      JAZ    E4              1          如果 ATOM(P)=0 则转移
E6    J2Z    DONE            n          E6. 往上
      ENT3   0,2             n-1        Q←T
      LDA    0,3(ATOM)       n-1
      JANZ   1F              n-1        如果 ATOM(T)=1 则转移
      LD2    0,3(BLINK)      t2         T←BLINK(Q)
      ST1    0,3(BLINK)      t2         BLINK(Q)←P
      ENT1   0,3             t2         P←Q
      JMP    E6              t2
1H    STZ    0,2(ATOM)       t1         ATOM(T)←0
      LD2    0,3(ALINK)      t1         T←ALINK(Q)
      ST1    0,3(ALINK)      t1         ALINK(Q)←P
      ENT1   0,3             t1         P←Q
E5    LD3    0,1(BLINK)      n          E5. 沿 BLINK 下降。Q←BLINK(P)
      J3Z    E6              n          如果 Q=Λ 则转移
      LDA    0,3             n-b2
      STX    0,3(MARK)       n-b2       MARK(Q)←1
      JANP   E6              n-b2       如果 NODE(Q)已被标记，则转移
      LDA    0,3(ATOM)       t2+a2
      JANZ   E6              t2+a2      如果 ATOM(Q)=1，则转移
      ST2    0,1(BLINK)      t2         BLINK(P)←T
E4A   ENT2   0,1             n-1        T←P
      ENT1   0,3             n-1        P←Q
E4    LD3    0,1(ALINK)      n          E4. 沿 ALINK 下降。Q←ALINK(P)
      J3Z    E5              n          如果 Q=Λ，则转移
```

LDA	0,3	$n-b_1$	
STX	0,3(MARK)	$n-b_1$	MARK(Q)←1
JANP	E5	$n-b_1$	如果 NODE(Q)已被标记,则转移
LDA	0,3(ATOM)	t_1+a_1	
JANZ	E5	t_1+a_1	如果 ATOM(Q) = 1,则转移
STX	0,1(ATOM)	t_1	ATOM(P)←1
ST2	0,1(ALINK)	t_1	ALINK(P)←T
JMP	E4A	t_1	T←P,P←Q,转到 E4。▌

由基尔霍夫定律,$t_1+t_2+1=n$。总的时间是$(34n+4t_1+3a-5b-8)u$,其中 n 是被标记的非原子节点个数,a 是被标记的原子个数,b 是在被标记的非原子节点中所遇到的 Λ 链接的个数,而 t_1 是我们沿 ALINK 下降的次数($0\leq t_1<n$)。

5.(下面是对于一级内存来说已知的最快的标记算法。)

S1. 置 MARK(P0)←1。如果 ATOM(P) = 1,则算法终止;否则置 S←0,R←P0,T←Λ。

S2. 置 P←BLINK(R)。如果 P = Λ 或 MARK(P) = 1,则转到 S3。否则置 MARK(P) = 1。现在如果 ATOM(P0) = 1,则转到 S3;否则如果 S < N 则置 S←S + 1,STACK[S] ←P,并转到 S3;否则转到 S5。

S3. 置 P←ALINK(R)。如果 P = Λ 或 MARK(P) = 1,则转到 S4。否则置 MARK(P)←1。现在如果 ATOM(P) = 1,则转到 S4;否则置 R←P 并返回 S2。

S4. 如果 S = 0,则终止本算法;否则置 R←STACK[S],S←S - 1,并转到 S2。

S5. 置 Q←ALINK(P)。如果 Q = Λ 或者 MARK(Q) = 1,则转到 S6。否则置 MARK(Q)←1。现在如果 ATOM(Q) = 1,则转到 S6;否则置 ATOM(P) ←1,ALINK(P)←T,T←P,P←Q,转到 S5。

S6. 置 Q←BLINK(P)。如果 Q = Λ 或者 MARK(Q) = 1,则转到 S7;否则置 MARK(Q) = 1。现在如果 ATOM(Q) = 1,则转到 S7;否则置 BLINK(P)←T,T←P,P←Q,转到 S5。

S7. 若 T = Λ,则转到 S3。否则置 Q←T。若 ATOM(Q) = 1,则置 ATOM(Q) ←0,T←ALINK(Q),ALINK(Q)←P, P←Q,并返回 S6。如果 ATOM(Q) = 0,则置 T←BLINK(Q),BLINK(Q)←P,P←Q,并返回 S7。▌

参考文献:*CACM* **10** (1967),501 ~ 506。

6. 来自废料收集的第二阶段(或许也包括初始阶段,如果所有的标记位这时皆置为 0 的话)。

7. 删去步骤 E2 和 E3,并删去 E4 中的"ATOM(P) ←1"。在步骤 E5 中置 MARK(P)←1,以及在步骤 E6 中用"MARK(Q) = 0","MARK(Q) = 1"分别代替当前的"ATOM(Q) = 1","ATOM(Q) = 0"。方法是仅在已经标记了左子树之后才来设置 MARK 位。这个算法甚至当树已经重叠(共享的)子树时也能进行工作;但是,它对于所有的递归列表结构,诸如以 NODE(ALINK(Q))作为 NODE(Q)之一祖宗的那些结构,不能工作。(注意一个已标记节点的 ALINK 从不改变。)

8. 解法 1:类似于算法 E,但更为简单。

F1. 置 T←Λ,P←P0。

F2. 置 MARK(P) ←1,并置 P←P + SIZE(P)。

F3. 如果 MARK(P) = 1,则转到 F5。

F4. 置 Q←LINK(P)。如果 Q≠Λ 而且 MARK(Q) = 0,则置 LINK(P)←T,T←P,P←Q,并转到 F2。否则置 P←P - 1 并返回 F3。

F5. 如果 T = Λ,则停止。否则置 Q←T,T←LINK(Q),LINK(Q)←P,P←Q - 1,并返回 F3。

一个类似的算法,它有时减轻内存的开销,而且它避免所有指向诸节点之中的指针,已经由

Lars-Erik Thorelli 提出,*BIT* **12** (1972),555 ~ 568。

解法 2:类似于算法 D。对于这一解法,我们假定 SIZE 字段大得足以包含一个链接地址。这样的假定对于问题的陈述可能是不正当的,但是它让我们使用比第一种解法稍快的方法,当其可应用时。

G1. 置 T←Λ,MARK(P0)←1,P←P0 + SIZE(P0)。

G2. 如果 MARK(P) = 1,则转到 G5。

G3. 置 Q←LINK(P),P←P - 1。

G4. 如果 Q≠Λ 且 MARK(Q) = 0,则置 MARK(Q)←1,S←SIZE(Q),SIZE(Q)←T,T←Q + S。返回到 G2。

G5. 如果 T = Λ,则停止。否则置 P←T 并求使得 MARK(Q) = 1 的 Q = P,P - 1,P - 2,…的第一个值;置 T←SIZE(Q)和 SIZE(Q)←P - Q。返回到 G2。 ▌

9. **H1.** 置 L←0,K←M + 1,MARK(0)←1,MARK(M + 1)←0。

H2. L 增加 1,而且如果 MARK(L) = 1,则重复这一步骤。

H3. K 减 1,而且如果 MARK(K) = 0,则重复这一步骤。

H4. 如果 L > K,则转到步骤 H5;否则置 NODE(L)←NODE(K),ALINK(K)←L,MARK(K)←0,并返回 H2。

H5. 对于 L = 1,2,…,K 做下列各项:置 MARK(L)←0。如果 ATOM(L) = 0 且 ALINK(L) > K,则置 ALINK(L)←ALINK(ALINK(L))。如果 ATOM(L) = 0 而且 BLINK(L) > K,则置 BLINK(L)←ALINK(BLINK(L))。 ▌

还请参见习题 2.5-33。

10. **Z1.** [初始化] 置 F←P0,R←AVAIL,NODE(R)⇐NODE(F),REF(F)←R。(这里 F 和 R 是在所遇到的所有头部节点的 REF 字段中建立起来的一个队列的指针。)

Z2. [开始新的列表] 置 P←F,Q←REF(P)。

Z3. [向右推进] 置 P←RLINK(P)。如果 P = Λ,则转到 Z6。

Z4. [复制一个节点] 置 Q1⇐AVAIL,RLINK(Q)←Q1,Q←Q1,NODE(Q)←NODE(P)。

Z5. [转换子列表链接] 如果T(P) = 1,则置 P1←REF(P),而且如果 REF(P1) = Λ 则置 REF(R)←P1,R⇐AVAIL,REF(P1)←R, NODE(R)←NODE(P1),REF(Q)←R。如果 T(P) = 1 而且 REF(P1)≠Λ,则置 REF(Q)←REF(P1)。转到 Z3。

Z6. [移到下一列表] 置 RLINK(Q)←Λ。如果 REF(F)≠R,则置 F←REF(REF(F))并返回 Z2。否则置 REF(R)←Λ,P←P0。

Z7. [最后的清扫] 置 Q←REF(P)。如果 Q≠Λ,则置 REF(P)←Λ 以及 P←Q 并重复步骤 Z7。 ▌

当然,REF 字段的这种用法使得不可能以算法 D 来进行废料收集;而且,算法 D 由于下述事实而被排斥,即在复制期间诸表尚未很好地形成。

对于列表的表示作相当弱的假定的许多漂亮的列表移动和列表复制的算法已被设计出来。参见 D. W. Clark, *CACM* **19** (1976),352 ~ 354;J. M. Robson, *CACM* **20** (1977),431 ~ 433。

11. 这里是一个用纸和笔的方法,它可以更加形式化地写出来回答这个问题:首先对于给定集合中的每个列表,附加一个惟一的名称(例如,一个大写字母);在这个例子中,我们可以有 $A = (a: C, b, a: F)$, $F = (b: D)$, $B = (a: F, b, a: E)$, $C = (b: G)$, $G = (a: C)$, $D = (a: F)$, $E =$

($b:G$)。现在构造必须证明其为相等的列表名称对偶的表。对于这个表逐次地加上列表对偶，直到或者是由于我们有一个在第一层上不一致的对偶(于是原来给定的列表是不相等的)，而发现矛盾为止，或者直到对偶的表不蕴涵任何更进一步的对偶(于是原来给定的列表是相等的)为止。在本例子中，对偶的这个表原来仅含有给定的对偶 AB；然后它得到进一步的对偶 CF, EF(通过匹配 A 和 B)，DG(从 CF)，而后我们有一个自一致的集合。

为了证明这个方法的正确性，我们来观察，(i)如果它回答"不相等"的答案，则给定的列表是不相等的；(ii)如果给定的列表是不相等的，则它返回"不相等"的答案；(iii)它总能终止。

12. 当 AVAIL 表含有 N 个节点时，这里 N 是在下面的讨论中有待选定的确定的常数，起动另一个与主程序共享计算机时间的共行程序，而且做下列工作：(a)在 AVAIL 表上标记所有 N 个节点；(b) 标记可为程序所访问的所有其它节点；(c) 把所有未标记的节点链接在一起以编制一个新的 AVAIL 表，供当前的 AVAIL 表为空时使用，以及(d)在所有节点中把标记位复位。人们必须选择 N 和共享的时间比例，以便保证，在从 AVAIL 表采用 N 个节点之前，完成(a)，(b)，(c)和(d)诸操作，同时主程序充分快地运行着。有必要在步骤(b)中多加用心，以确保当程序继续运行时把所有"可为程序访问"的节点都包括进来；这里省略了细节。如果在(c)中形成的表有少于 N 个的节点，则由于内存空间可能穷尽了，可能有必要最终停下来。[关于进一步的信息，见 Guy L. Steele Jr., *CACM* **18** (1975), 495 ~ 508; P. Wadler, *CACM* **19** (1976), 495 ~ 500; E. W. Dijkstra, L. Lamport, A. J. Martin, C. S. Scholten 及 E. F. M. Steffens, *CACM* **21** (1978), 966 ~ 975; H. G. Baker, Jr., *CACM* **21** (1978), 280 ~ 294。]

2.4 节

1. 先根序。

2. 实际上与所建立的 Data Table(数据表)的条目数成比例。

3. 把步骤 A5 改变成：

A5′. [撤消顶层] 撤消顶层上栈条目；而且如果在栈顶上的新的层号 ≥ L，则命(L1, P1)是在栈顶上的新的条目，而且重复这一步骤。否则置 SIB(P1)←Q 然后命(L1, P1)为在栈顶上的新条目。

4.(由 David S. Wise 所提供的解) 规则(c)被违反，当且仅当有一个数据项，它的完备的定性 A_0 OF⋯OF A_n 也是对某个其它数据项的一个 COBOL 访问。由于父亲 A_1 OF⋯OF A_n 也必定满足规则(c)，因此我们可以假定这其它数据项是相同父亲的一个后裔。因此算法 A 将被扩充来校验，随着把每个新数据加到 Data Table，它的父亲是否为相同名字的任何其它项的父亲，或者相同名字的任何其它项的父亲是否在栈中。(当父亲是 Λ 时，它是每个人的祖宗而且总在栈上。)

另一方面，如果我们保留算法 A 原来的样子，则当试图使用非法的项时，COBOL 程序员将从算法 B 获得错误的消息。只有 MOVE CORRESPONDING 能够无错误地利用这样的项。

5. 作下列的改动：

步骤	被代替者	代替者
B1.	P←LINK(P_0)	P←LINK(INFO(T))
B2.	k←0	K←T
B3.	$k < n$	RLINK(K) ≠ Λ
B4.	k←$k+1$	K←RLINK(K)
B6.	NAME(S) = P_k	NAME(S) = INFO(K)

6. 算法 B 的一个简单的修改，使得它仅仅寻找完备的访问(如果在步骤 B3 中，$k = n$ 且 PAR-

ENT(S) ≠Λ,或者如果在步骤 B6 中 NAME(S) ≠ P_k,则置 P←PREV(P)并转到 B2)。想法是首先完整运行这个修改的算法 B;然后,如果 Q 仍然是 Λ,则实施未修改的算法。

7. MOVE MONTH OF DATE OF SALES TO MONTH OF DATE OF PURCHASES。MOVE DAY OF DATE OF SALES TO DAY OF DATE OF PURCHASES。MOVE YEAR OF DATE OF SALES TO YEAR OF DATE OF PURCHASES。MOVE ITEM OF TRANSACTION OF SALES TO ITEM OF TRANSACTION OF PURCHASES。MOVE QUANTITY OF TRANSACTION OF SALES TO QUANTITY OF TRANSACTION OF PURCHASES。MOVE PRICE OF TRANSACTION OF SALES TO PRICE OF TRANSACTION OF PURCHASES。MOVE TAX OF TRANSACTION OF SALES TO TAX OF TRANSACTION OF PURCHASES。

8. 当且仅当 α 和 β 是一个初等项。(在作者关于算法 C 的第一稿中,没有适当地处理这一情况,实际上反而把这算法弄得更复杂,也许说明这一点有所裨益。)

9. 如果 α 和 β 都不是初等的,则“MOVE CORRESPONDING α TO β”,等价于取遍对于分组 α 和 β 共同的所有名称 A 的语句“MOVE CORRESPONDING A OF α TO A OF β”之集合。(比起正文中所给出的更为传统也更为繁琐的“MOVE CORRESPONDING”的定义来说,这是更优雅的叙述定义的方式。)利用归纳法证明,步骤 C2 到 C5 将最终以 P = P0 和 Q = Q0 终止,我们可以验证算法 C 满足这个定义。关于这个证明的进一步的细节,可以像我们在“树的归纳法”中多次做过的那样填充(参照算法 2.3.1T 的证明)。

10. (a)置 S1←LINK(P_k)。然后重复地置 S1←PREV(S1) 零次或多次直到或 S1 = Λ(NAME(S)≠ P_k)或 S1 = S (NAME(S) = P_k)为止。(b) 置 P1←P 然后置 P1←PREV(P1)零次或多次直到 PREV(P1) = Λ 为止;对于变量 Q1 和 Q 进行类似的操作,而后测试是否 P1 = Q1。或者,如果 Data Table 的条目是有序的,使得对于所有的 P,PREV(P) < P,则依赖于是否 P > Q,以及沿着较大的条目的 PREV 链来看看是否可遇到较小的条目,就可以以一种明显的方式来进行更快的测试。

11. 通过增加新的链接字段 SIB1(P)≡CHILD(PARENT(P)),在步骤 C4 的速度方面将实现一点点改进。更有意义的是,我们可以修改 CHILD 和 SIB 的链接使得 NAME(SIB(P)) > NAME(P);这将相当大地加速步骤 C3 中的检索,因为它仅仅需要扫描每个家族一次来寻找匹配的成员。因此,这个改动将撤消出现在算法 C 中的仅有的“查找”。对于这样的解释,算法 A 和 C 很容易加以修改,而且读者可能会发现这是一个有趣的习题。(然而,如果我们考虑 MOVE CORRESPONDING 语句的相对频率以及家族类的通常大小,则在真正的 COBOL 程序的翻译中,得到的加速将不是特别有意义的。)

12. 保持步骤 B1,B2,B3 不变;改变其它步骤如下:

B4. 置 $k \leftarrow k+1$,R←LINK(P_k)。

B5. 如果 R = Λ,则没有匹配;置 P←PREV(P)并转到 B2。如果 R < S≤SCOPE(R),则置 S←R 并转到 B3。否则置 R←PREV(R)并重复步骤 B5。▌

这一算法不适合于习题 6 的 PL/I 的约定。

13. 使用相同的算法,减去置 NAME,PARENT,CHILD 以及 SIB 诸操作。每当撤消步骤 A5 中的栈顶的条目时,就置 SCOPE(P1)←Q - 1。当步骤 A2 中的输入穷尽时,简单地置 L←0,并继续,然后如果在步骤 A7 中 L = 0,则这算法结束。

14. 下列算法使用了辅助栈,其编了号的步骤与正文的算法直接对应。

C1. 置 P←P0,Q←Q0,并置栈的内容为空。

C2. 如果 SCOPE(P) = P 或 SCOPE(Q) = Q,则输出(P,Q)作为所求的对偶之一并转到 C5。否

则把(P,Q)放入栈并置 P←P+1,Q←Q+1。

C3. 确定 P 和 Q 是否指向具有相同名称的条目(参照习题 10(b))。如果是,则转到 C2。如果不是,则命(P1,Q1)是栈顶上的条目;如果 SCOPE(Q) < SCOPE(Q1),则置 Q←SCOPE(Q)+1并重复步骤 C3。

C4. 命(P1,Q1)是栈顶上的条目。如果 SCOPE(P) < SCOPE(P1),则置 P←SCOPE(P)+1,Q←Q1+1 并且转回到 C3。如果 SCOPE(P) = SCOPE(P1),则置 P←P1,Q←Q1,并且撤消栈顶上的条目。

C5. 如果栈为空,则算法结束。否则转到 C4。▎

2.5 节

1. 在这种意外的环境下,可以使用类似栈的操作如下:设内存池区域是单元 0 到 M-1,并设 AVAIL 指向最低的自由单元。为保留 N 个字,如果 AVAIL+N≥M,则报告失败,否则置 AVAIL←AVAIL+N。为释放这 N 个字,只须置 AVAIL←AVAIL-N 即可。

类似地,循环的类似队列的操作适宜于先进先出的规则。

2. 对于长度为 l 的项,存储空间的量是 $k\lceil l/(k-b)\rceil$,它有平均值 $kL/(k-b)+(1-\alpha)k$,其中假定 α 是 1/2,与 k 无关。当 $k=b+\sqrt{2bL}$ 时,这个表达式取极小值(对于 k 的实数值而言)。所以选择 k 为刚好大于或刚好小于这个值的一个整数,无论哪个总给出 $kL/(k-b)+\frac{1}{2}k$ 的最低值。例如,如果 $b=1$ 和 $L=10$,则 $k\approx 1+\sqrt{20}=5$ 或 6;两者都同样好。关于这问题的大量细节,请见*JACM* **12** (1965),53~70。

4. rI1≡Q, rI2≡P。

```
A1     LDA     N            rA←N
       ENT2    AVAIL        P←LOC(AVAIL)
A2A    ENT1    0,2          Q←P
A2     LD2     0,1(LINK)    P←LINK(Q)
       J2N     OVERFLOW     如果 P=Λ,则无单元
A3     CMPA    0,2(SIZE)
       JG      A2A          如果 N>SIZE(P),则转移
A4     SUB     0,2(SIZE)    rA←N-SIZE(P)≡K
       JANZ    *+3          如果 K≠0 则转移
       LDX     0,2(LINK)
       STX     0,1(LINK)    LINK(Q)←LINK(P)
       STA     0,2(SIZE)    SIZE(P)←K
       LD1     0,2(SIZE)    可选的结束
       INC1    0,2          置 rI1←P+K ▎
```

5. 大概不是。刚好在单元 P 之前的不可用存储区域,将随即变成可用的,而且它的长度将增加数量 K;99 的增量将是不可忽视的。

6. 想法是每次在 AVAIL 表的不同部分试行检索。我们可以使用“遨游指针”,例如称它为 ROVER,对它处理如下:在步骤 A1,置 Q←ROVER。在步骤 A4 之后若 LINK(Q)≠Λ 则置 ROVER←LINK(Q)否则置 ROVER←LOC(AVAIL)。在步骤 A2,在算法 A 的一个特定执行期间当头一次 P=Λ 时,置 Q←LOC(AVAIL)并重复步骤 A2。当第二次 P=Λ 时,这个算法将以失败而结束。在这

种方式下，ROVER 将趋于指向 AVAIL 表中的随机的一点，而且大小将更为平衡。在程序开始时，置 ROVER←LOC(AVAIL)；在程序中别的地方也有必要置 ROVER 成为 LOC(AVAIL)，在这些地方块是由 AVAIL 表中取出的，而块的地址等于 ROVER 之当前设定。

7. 对于大小 800，1300 的内存请求，可用内存区域为 2000，1000。

[最坏适合获得成功，而最好适合反倒失败的这样的一个例子，已经由 R. J. Weiland 构造出来。]

8. 在步骤 A1，还置 M←∞，R←Λ。在步骤 A2，如果 P = Λ 则转到 A6。在步骤 A3，转到 A5 而不是 A4。加上新的步骤如下：

A5. [较好适合？] 如果 M > SIZE(P)，则置 R←Q 和 M←SIZE(P)。然后置 Q←P 并返回 A2。

A6. [找到一个？] 如果 R = Λ，则本算法以失败结束。否则置 Q←R，P←LINK(Q)并转到 A4。 ▌

9. 显然，如果我们如此幸运地找出 SIZE(P) = N，则我们有最好适合，而且不需要进行进一步的查找。(当仅有很少不同的块大小时，这颇为经常地出现。)如果正使用的是类似于算法 C 中的"边界标记方法"，则有可能以按大小排序的顺序来维持这 AVAIL 表，所以平均说来，查找的长度可能减少到这表长度的一半以至更少。但最好的解法是如同在 6.2.3 小节中所述的那样，把 AVAIL 表造入平衡的树结构中，如果预期它是很长的话。

10. 作如下的变动：

在步骤 B3 开始插入"如果 P0 + N > P 和 P≠Λ，则置 N←max(N, P + SIZE(P) − P0)，P←LINK(P)并重复步骤 B3。"

步骤 B4，将"Q + SIZE(Q) = P0"改作"Q + SIZE(Q) ≥ P0"；而且将"SIZE(Q)←SIZE(Q) + N"改作"SIZE(Q)←max(SIZE(Q), P0 + N − Q)"。

11. 如果 P0 大于 ROVER，则在步骤 B1 中我们可以置 Q←ROVER 以代替 Q←LOC(AVAIL)。如果在 AVAIL 表中有 n 个条目，则步骤 B2 的迭代的平均数是 $(2n+3)(n+2)/6(n+1) = n/3 + 5/6 + O(1/n)$。例如如果 $n = 2$，则我们得到 9 种同等可能的情况，其中 P1 和 P2 指向两个现存的可用块：

	P0 < P1	P1 < P0 < P2	P2 < P0
ROVER = P1	1	1	2
ROVER = P2	1	2	1
ROVER = LOC(AVAIL)	1	2	3

这个图表说明了在每种情况下所需要的迭代数。平均是

$$\frac{1}{9}\left(\binom{2}{2}+\binom{3}{2}+\binom{4}{2}+\binom{3}{2}+\binom{2}{2}\right) = \frac{1}{9}\left(\binom{5}{3}+\binom{4}{3}\right) = \frac{14}{9}$$

12. **A1.** 置 P←ROVER，F←0。

A2. 如果 P = LOC(AVAIL)而且 F = 0，则置 P←AVAIL，F←1 并重复步骤 A2。如果 P = LOC(AVAIL)而且 F≠0，则算法以失败而结束。

A3. 如果 SIZE(P) ≥N，则转到 A4；否则置 P←LINK(P)并返回 A2。

A4. 置 ROVER←LINK(P)，K←SIZE(P) − N。如果 K < c(其中 $c \geq 2$ 是一个常数)，则置 LINK(LINK(P + 1))←ROVER，LINK(ROVER + 1) ←LINK(P + 1)，L←P；否则置 L←P + K，SIZE(P)←SIZE(L − 1) ←K，TAG(L − 1) ←" − "，SIZE(L)←N。最后置 TAG(L) ←TAG(L + SIZE(L) − 1) ←" + "。

13. rI1≡P, rX≡F, rI2≡L

```
LINK   EQU    4:5
SIZE   EQU    1:2
TSIZE  EQU    0:2
TAG    EQU    0:0
A1     LDA    N                  rA←N
       SLA    3                  移入 SIZE 字段
       ENTX   0                  F←0
       LD1    ROVER              P←ROVER
       JMP    A2
A3     CMPA   0,1(SIZE)
       JLE    A4                 如果 N≤SIZE(P),则转移
       LD1    0,1(LINK)          P←LINK(P)
A2     ENT2   -AVAIL,1           rI2←P-LOC(AVAIL)
       J2NZ   A3
       JXNZ   OVERFOLW           F≠0?
       ENTX   1                  置 F←1
       LD1    AVAIL(LINK)        P←AVAIL
       JMP    A2
A4     LD2    0,1(LINK)
       ST2    ROVER              ROVER←LINK(P)
       LDA    0,1(SIZE)          rA≡K←SIZE(P)-N
       SUB    N
       CMPA   =c=
       JGE    1F                 如果 K≥c,则转移
       LD3    1,1(LINK)          rI3←LINK(P+1)
       ST2    0,3(LINK)          LINK(rI3)←ROVER
       ST3    1,2(LINK)          LINK(ROVER+1)←rI3
       ENT2   0,1                L←P
       LD3    0,1(SIZE)          rI3←SIZE(P)
       JMP    2F
1H     STA    0,1(SIZE)          SIZE(P)←K
       LD2    0,1(SIZE)
       INC2   0,1                L←P+K
       LDAN   0,1(SIZE)          rA←-K
       STA    -1,2(TSIZE)        SIZE(L-1)←K,TAG(L-1)←"-"
       LD3    N                  rI3←N
2H     ST3    0,2(TSIZE)         TAG(L)←"+",还置 SIZE(L)←rI3
       INC3   0,2
       STZ    -1,3(TAG)          TAG(L+SIZE(L)-1)←"+"  ▮
```

14.(a)在步骤 C2 中,需要这个字段来定位块的开始。它可以用对于此块的第一个字的链接来加以代替(也许更有利)。(b)由于有时有必要保留 N 个以上的字(例如,如果 K = 1),因而需要这个字段,而且,当这个块随后被释放时,必须知道保留的数量。

15,16. rI1 ≡ P0, rI2 ≡ P1, rI3 ≡ F, rI4 ≡ B, rI6 ≡ − N。

```
D1   LD1    P0            D1.
     LD2    0,1(SIZE)
     ENN6   0,2           N←SIZE(P0)
     INC2   0,1           P1←P0 + N
     LD5    0,2(TSIZE)
     J5N    D4            如果 TAG(P1) = "－",则转 D4
D2   LD5    -1,1(TSIZE)   D2.
     J5N    D7            如果 TAG(P0 - 1) = "－",则转 D7
D3   LD3    AVAIL(LINK)   D3. 置 F←AVAIL
     ENT4   AVAIL         B←LOC(AVAIL)
     JMP    D5            转 D5
D4   INC6   0,5           D4. N←N + SIZE(P1)
     LD3    0,2(LINK)     F←LINK(P1)
     LD4    1,2(LINK)     B←LINK(P1 + 1)
     CMP2   ROVER         (新的代码,由于
     JNE    *+3             习题 12 的 ROVER 的特性:
     ENTX   AVAIL           如果 P1 = ROVER,则
     STX    ROVER             置 ROVER←LOC(AVAIL))
     DEC2   0,5           P1←P1 + SIZE(P1)
     LD5    -1,1(TSIZE)
     J5N    D6            如果 TAG(P0 - 1) = "－",则转 D6
D5   ST3    0,1(LINK)     D5. LINK(P0)←F
     ST4    1,1(LINK)     LINK(P0 + 1) ←B
     ST1    1,3(LINK)     LINK(F + 1) ←P0
     ST1    0,4(LINK)     LINK(B) ←P0
     JMP    D8            转 D8
D6   ST3    0,4(LINK)     D6. LINK(B) ←F
     ST4    1,3(LINK)     LINK(F + 1) ←B
D7   INC6   0,5           D7. N←N + SIZE(P0 - 1)
     INC1   0,5           P0←P0 - SIZE(P0 - 1)
D8   ST6    0,1(TSIZE)    D8. SIZE(P0) ←N, TAG(P0) ←"－"
     ST6    -1,2(TSIZE)   SIZE(P1 - 1) ←N, TAG(P1 - 1) ←"－"  ▌
```

17. 两个 LINK 字段都等于 LOC(AVAIL)。

18. 算法 A 保留一个大块的高端。当存储完全可用时,首先适合的方法实际上是通过保留高阶位置开始的,但一旦这些变成再次可用时,它们就不是重新保留的,因为"适合"通常已经在较低位置中找到;于是在内存低端的初始大块,很快地消失于"首先适合"。然而大的块很少是

"最好适合"的，所以在内存的开始，最好适合的方法保留一个大的块。

19. 利用习题 12 的算法，但从步骤 A4 删去对 SIZE(L－1)，TAG(L－1)及 TAG(L＋SIZE(L)－1)的访问；并且在步骤 A2 和 A3 之间插入下列诸动作：

A2 $\frac{1}{2}$. 置 P1←P＋SIZE(P)。如果 TAG(P1)＝"＋"，则进行到 A3。否则置 P2←LINK(P1)，LINK(P2＋1)←LINK(P1＋1)，LINK(LINK(P1＋1))←P2，SIZE(P)←SIZE(P)＋SIZE(P1)。如果 ROVER＝P1 则置 ROVER←P2，重复 A2 $\frac{1}{2}$。

显然(2)，(3)，(4)的情况这里不能出现；对存储分配惟一的实际效果是，这里的查找将趋于比习题 12 中的更长，而且有时 K 将小于 c，尽管实际上在这个块前面还有另一个我们还不知道的可用块。

(一种选择是解除内循环 A3，而且仅仅在步骤 A4 中，在最后的分配之前或者在内循环中，当算法否则就将以失败结束时，来进行解除。这个选择要求进行模拟研究，来看看它是否是有所改进。)

[通过一些改进，这个算法在 $\mathrm{T_EX}$ 和 METAFONT 的实现中已经证明是十分令人满意的。参见 *T_EX: The Program* (Addison-Wesley, 1986), § 125。]

20. 当在解除循环期间，发现了伙伴可利用时，我们要把此块从它的 AVAIL[k]表中撤消，但我们不知道修改哪一些链接，除非(i)我们作了可能很长的查找，或者(ii)该表是双重链接的。

21. 如果 $n=2^k\alpha$，其中 $1\leqslant\alpha\leqslant 2$，则 a_n 是 $2^{2k+1}(\alpha-2/3)+1/3$，而 b_n 是 $2^{2k-1}\alpha^2+2^{k-1}\alpha$。对于很大的 n，比率 a_n/b_n 实际上是 $4(\alpha-2/3)/\alpha^2$，当 $\alpha=1$ 和 2 时它取其极小值 4/3，而当 $\alpha=1\frac{1}{3}$ 时取其极大值 3/2。所以 a_n/b_n 无极限，它在这两个极端之间振荡。然而，4.2.4 小节的取平均方法确实产生 $4(\ln 2)^{-1}\int_1^2(\alpha-\frac{2}{3})\mathrm{d}\alpha/\alpha^3=(\ln 2)^{-1}\approx 1.44$ 的平均比率。

22. 这个想法要求在 11 字块的若干字中，有一个 TAG 字段，而不仅在第一个字中。这是一个切实可行的想法，如果这些额外的 TAG 位可以抽出的话，而且它显得特别适合于在计算机硬件中使用。

23. 011011110100；011011100000。

24. 这在程序中引进了一个错误；当 TAG(0)＝1 时我们可能到达步骤 S1；因为 S2 可能返回到 S1。为使它有效，在步骤 S2 中，在"L←P"之后，加上"TAG(L)←0"。(换成假定 TAG(2^m)＝0 更为容易。)

25. 这一想法绝对正确。(批评未必就是否定。)对于 $n<k\leqslant m$，表头 AVAIL[k]可以被消去；如果在步骤 R1，S1 中把"m"改变成"n"，则可使用正文中的算法。初始条件(13)，(14)应该改变成指出大小为 2^n 的 2^{m-n} 个块，以替代大小为 2^m 的一个块。

26. 使用 M 的二进制表示，我们可容易地修改初始条件(13)，(14)，使得所有的内存单元被划分成大小为 2 的乘幂的一些块，而且这些块处于大小递减的顺序。在算法 S 中，每当 P≥M－2^k 时，TAG(P)就应认为是 0。

27. rI1≡k，rI2≡j，rI3≡$j-k$，rI4≡L，LOC(AVAIL[j])＝AVAIL＋j；假定对于 $0\leqslant j\leqslant m$，有一辅助表格 TWO[j]＝2^j，存于单元 TWO＋j 中。进一步假定"＋"和"－"表示 0 和 1 的标记，而且 TAG(LOC(AVAIL[j]))＝"－"；但 TAG(LOC(AVAIL[$m+1$]))＝"＋"是一个标记。

```
00  KVAL  EQU   5:5
01  TAG   EQU   0:0
```

```
LINKF EQU    1:2
LINKB EQU    3:4
TLNKF EQU    0:2
R1    LD1    K                 1        R1. 找块
      ENT2   0,1               1        j←k
      ENT3   0                 1
      LD4    AVAIL,2(LINKF)    1
1H    ENT5   AVAIL,2           1+R
      DEC5   0,4               1+R
      J5NZ   R2                1+R      若AVAILF[j]≠LOC(AVAIL[j])则转移
      INC2   1                 R        增加 j
      INC3   1                 R
      LD4N   AVAIL,2 (TLNKF)   R
      J4NN   1B                R        j≤m 吗?
      JMP    OVERFLOW
R2    LD5    0,4(LINKF)        1        R2. 从表中撤消
      ST5    AVAIL,2(LINKF)    1        AVAILF[j]←LINKF(L)
      ENTA   AVAIL,2           1
      STA    0,5(LINKB)        1        LINKB(L)← LOC(AVAIL[j])
      STZ    0,4(TAG)          1        TAG(L)←0
R3    J3Z    DONE              1        R3. 需要分开吗?
R4    DEC3   1                 R        R4. 分开
      DEC2   1                 R        减小 j
      LD5    TWO,2             R        rI5≡P
      INC5   0,4               R        P←L+2^j
      ENNA   AVAIL,2           R
      STA    0,5(TLNKF)        R        TAG(P)←1,LINKF(P)←LOC(AVAIL[j])
      STA    0,5(LINKB)        R        LINKB(P)←LOC(AVAIL[j])
      ST5    AVAIL,2(LINKF)    R        AVAILF[j]←P
      ST5    AVAIL,2(LINKB)    R        AVAILB[j]←P
      ST2    0,5(KVAL)         R        KVAL(P)←j
      J3P    R4                R        转到 R3
      DONE   ···                          ▌
```

28. $rI1\equiv k$，$rI5\equiv$ P，$rI4\equiv$ L；假定 TAG(2^m) = "+"。

```
S1    LD4    L                 1        S1.是可利用的伙伴吗?
      LD1    K                 1
1H    ENTA   0,4               1+S
      XOR    TWO,1             1+S      rA←buddy_k(L)
      STA    TEMP              1+S
      LD5    TEMP              1+S      P←rA
```

```
        LDA    0,5              1+S
        JANN   S3               1+S    如果 TAG(P)=0 则转移
        CMP1   0,5(KVAL)        B+S
        JNE    S3               B+S    如果 KVAL(P)≠k 则转移
S2      LD2    0,5(LINKB)       S      S2. 与伙伴组合
        LD3    0,5(LINKF)       S
        ST3    0,2(LINKF)       S      LINKF(LINKB(P))←LINKF(P)
        ST2    0,3(LINKB)       S      LINKF(LINKF(P))←LINKB(P)
        INC1   1                S      增加 k
        CMP4   TEMP             S
        JL     1B               S
        ENT4   0,5              A      如果 L>P,则置 L←P
        JMP    1B               A
S3      LD2    AVAIL,1(LINKF)   1      S3. 置入表中
        ENNA   AVAIL,1          1
        STA    0,4(0:4)         1      TAG(L)←1,LINKB(L)←LOC(AVAIL[k])
        ST2    0,4(LINKF)       1      LINKF(L)←AVAILF[k]
        ST1    0,4(KVAL)        1      KVAL(L)←k
        ST4    0,2(LINKB)       1      LINKB(AVAILF[k])←L
        ST4    AVAIL,1(LINKF)   1      AVAILF[k]←L  ▌
```

29. 是的,但仅仅以牺牲某个检索为代价,或者(更好地)想办法组装 TAG 二进位的附加表格。(谨建议在算法 S 的运行期间不把伙伴联合到一起,而仅仅在算法 R 中这样做,如果没有足够大的块区满足这个要求的话;但这大概会导致很坏的内存碎片。)

31. 参见 David. L. Russell,*SICOMP* **6** (1977),607 ~ 621。

33. **G1.** [清除诸 LINK] 置 P←1,并重复操作 LINK(P)←Λ,P←P + SIZE(P)直到 P = AVAIL 为止。(这仅仅在每个节点的第一个字中,把 LINK 字段置成 Λ;在大数情况下,我们可以假定,这个步骤是不必要的,因为 LINK(P)在以下的步骤 G9 中置成 Λ,并可通过存储分配程序置成 Λ。)

G2. [初始化标记阶段] 置 TOP←USE,LINK(TOP)←AVAIL,LINK(AVAIL)←Λ。(如同在算法 2.3.5D 中那样,TOP 指向栈的顶部。)

G3. [弹出栈] 置 P←TOP,TOP←LINK(TOP)。如果 TOP = Λ,则转到 G5。

G4. [放置新链接到栈] 对于 $1 \le k \le$ T(P),进行下列操作:置 Q←LINK(P + k);然后如果 Q≠Λ 且 LINK(Q) = Λ,则置 LINK(Q)←TOP,TOP←Q。然后返回到 G3。

G5. [初始化下一阶段] (现在 P = AVAIL,而且加标记阶段已经完成,使得每个可访问的节点的第一个字,都有一非空的 LINK。我们的下一个目标是为了加快后面步骤的速度,把相邻的不可访问的节点组合起来。而且对可访问节点指定新的地址。)置 Q←1,LINK(AVAIL)←Q,SIZE(AVAIL)←0,P←1。(单元 AVAIL 正用作一个标志,以标记在随后的阶段中一个循环之结束。)

G6. [指定新的地址] 如果 LINK(P) = Λ,则转到 G7。否则如果 SIZE(P) = 0,则转到 G8。否则置 LINK(P)←Q,Q←Q + SIZE(P),P←P + SIZE(P),并重复这一步骤。

G7. [折叠可用区域] 如果 LINK(P + SIZE(P)) = Λ,则以 SIZE(P + SIZE(P))增加 SIZE(P)以并重复这--步骤。否则置P←P + SIZE(P)并返回 G6。

G8. [转换全部链接] (现在每个可访问的节点的第一个字中的 LINK 字段,都包含着节点将被移动到的地址。)置 USE←LINK(USE),以及 AVAIL←Q。然后置 P←1,并重复下列操作直到 SIZE(P) = 0 为止:如果 LINK(P)≠Λ,则对于所有使 P < Q≤P + T(P)和 LINK(Q)≠Λ 的 Q 置 LINK(Q)←LINK(LINK(Q));然后不管 LINK(P)的值是什么,都置 P←P + SIZE(P)。

G9. [移动] 置 P←1,并且重复下列操作直到 SIZE(P) = 0 为止:置 Q←LINK(P),而且如果 Q≠Λ,则置 LINK(P)←Λ 以及 NODE(Q)←NODE(P);然后无论 Q = Λ 与否,置 P←P + SIZE(P)。(操作 NODE(Q)←NODE(P)意味着移动 SIZE(P)个字;我们总有 Q≤P,所以按从最小单元到最大单元的顺序移动这些字是安全的。) ▌

[这个算法称为"LISP 2 废料收集程序"。还有一个有趣的选择,它不要求在节点开始处的 LINK字段,而是基于把指向每个节点的所有指针都链接在一起的思想。——请见 Lars-Erik Thorelli,*BIT* **16** (1976),426 ~ 441; R. B. K. Dewar 和 A. P. McCann, *Software Practice & Exp.* **7** (1977),95 ~ 113;F. Lockwood Morris, *CACM* **21** (1978),662 ~ 665,**22** (1979),571; H. B. M. Jonkers, *Inf. Proc. Letters* **9** (1979),26 ~ 30;J. J. Martin,*CACM* **25** (1982),571 ~ 581;F. Lockwood Morris, *Inf. Proc. Letters* **15** (1982),139 ~ 142,**16** (1983),215。其它方法已发表于 B.K.Haddon 和 W.M.Waite, *Comp. J.* **10** (1967),162 ~ 165;B. Wegbreit,*Comp. J.* **15** (1972),204 ~ 208;D.A.Zave,*Inf. Proc. Letters* **3** (1975),167 ~ 169。Cohen 和 Nicolau 在 *ACM Trans. Prog. Languages and Systems* **5** (1983),532 ~ 553 上对这些方法中的四个进行了分析。]

34. 设 TOP≡rI1,Q≡rI2,P≡rI3,k≡rI4,SIZE(P)≡rI5。进一步假定 Λ = 0 和 LINK(0)≠0 以简化步骤 G4。省略去步骤 G1。

```
LINK  EQU   4:5
INFO  EQU   0:3
SIZE  EQU   1:2
T     EQU   3:3
G2    LD1   USE         1       G2. 初始化标记阶段。TOP←USE
      LD2   AVAIL       1
      ST2   0,1(LINK)   1       LINK(TOP) ←AVAIL
      STZ   0,2(LINK)   1       LINK(AVAIL) ←Λ
G3    ENT3  0,1         a + 1   G3. 弹出栈。P←TOP
      LD1   0,1(LINK)   a + 1   TOP←LINK(TOP)
      J1Z   G5          a + 1   如果 TOP = Λ,则转到 G5
G4    LD4   0,3(T)      a       G4. 把新链接放入栈。k←T(P)
1H    J4Z   G3          a + b   k = 0?
      INC3  1           b       P←P + 1
      DEC4  1           b       k←k - 1
      LD2   0,3(LINK)   b       Q←LINK(P)
      LDA   0,2(LINK)   b
      JANZ  1B          b       如果 LINK(Q)≠Λ,则转移
      ST1   0,2(LINK)   a - 1   否则置 LINK(Q) ←TOP
```

```
         ENT1  0,2         a-1        TOP←Q
         JMP   1B          a-1
G5       ENT2  1           1          G5. 初始化下一阶段。Q←1
         ST2   0,3         1          LINK(AVAIL) ←1,SIZE(AVAIL) ←0
         ENT3  1           1          P←1
         JMP   G6          1
1H       ST2   0,3(LINK)   a          LINK(P)←Q
         INC2  0,5         a          Q←Q+SIZE(P)
         INC3  0,5         a          P←P+SIZE(P)
G6       LDA   0,3(LINK)   a+1        G6. 指定新地址
G6A      LD5   0,3(SIZE)   a+c+1
         JAZ   G7          a+c+1      如果 LINK(P)=Λ,则转移
         J5NZ  1B          a+1        如果 SIZE(P) ≠0,则转移
G8       LD1   USE         1          G8. 转换全部链接
         LDA   0,1(LINK)   1
         STA   USE         1          USE←LINK(USE)
         ST2   AVAIL       1          AVAIL←Q
         ENT3  1           1          P←1
         JMP   G8P         1
1H       LD6   0,6(SIZE)   d
         INC5  0,6         d          rI5←rI5+SIZE(P+SIZE(P))
G7       ENT6  0,3         c+d        G7. 折叠可用区域
         INC6  0,5         c+d        rI6←P+SIZE(P)
         LDA   0,6(LINK)   c+d
         JAZ   1B          c+d        如果 LINK(rI6)=Λ,则转移
         ST5   0,3(SIZE)   c          SIZE(P)←rI5
         INC3  0,5         c          P←P+SIZE(P)
         JMP   G6A         c
2H       DEC4  1           b          k←k-1
         INC2  1           b          Q←Q+1
         LD6   0,2(LINK)   b
         LDA   0,6(LINK)   b
         STA   0,2(LINK)   b          LINK(Q) ←LINK(LINK(Q))
1H       J4NZ  2B          a+b        如果 k≠0,则转移
3H       INC3  0,5         a+b        P←P+SIZE(P)
G8P      LDA   0,3(LINK)   1+a+c
         LD5   0,3(SIZE)   1+a+c
         JAZ   3B          1+a+c      LINK(P)=Λ 吗?
         LD4   0,3(T)      1+a        k←T(P)
         ENT2  0,3         1+a        Q←P
         J5NZ  1B          1+a        转移,除非 SIZE(P)=0
```

```
61  G9    ENT3   1           1          G9. 移动。P←1
62        ENT1   1           1          为 MOVE 指令置 rI1
63        JMP    G9P         1
64  1H    ST2    0,3(LINK)   a          LINK(P)←Λ
65        ST5    *+1(4:4)    a
66        MOVE   0,3(*)      a          NODE(rI1)←NODE(P),rI1←rI1 + SIZE(P)
67  3H    INC3   0,5         a + c      P←P + SIZE(P)
68  G9P   LDA    0,3(LINK)   1 + a + c
69        LD5    0,3(SIZE)   1 + a + c
70        JAZ    3B          1 + a + c  如果 LINK(P) = Λ,则转移
71        J5NZ   1B          1 + a      转移,除非 SIZE(P) = 0 ▌
```

在行 66 中,我们假定每个节点的大小都充分小,使得它能以一单条 MOVE 指令来进行移动;对于大多数情况说来,当这种类型的废料收集可应用时,这似乎是一个公正的假定。

这个程序的总运行时间为$(44a + 17b + 2w + 25c + 8d + 47)u$,其中 a 是可访问的节点数,b 是其中链接字段的个数,c 是其前边没有不可访问节点领先的那些不可访问节点的个数,d 是其前边有不可访问节点的那些不可访问节点的个数,而 w 是在可访问节点中的总字数。如果内存含有 n 个节点,其中有 ρn 个是不可访问的,则我们可估计 $a = (1 - \rho)n$, $c = (1 - \rho)\rho n$, $d = \rho^2 n$。例:5 字的节点(平均地说),每个节点有两个链接字段(平均地说),以及 1000 个节点的内存。则当 $\rho = 1/5$ 时,恢复每个可用节点花费 $374u$;当 $\rho = 1/2$ 时,花费 $104u$;而当 $\rho = 4/5$ 时,仅花 $33u$。

36. 单个顾客将可能在 16 个座席 1,3,4,6,⋯,23 中之一入座。如果进来一对顾客,则对他俩来说必定有位置,否则至少有两个人在座席(1,2,3),至少两个在(4,5,6),⋯,至少两个在(19,20,21),以及至少一人在 22 或 23,所以至少有 15 人已入座。

37. 首先 16 个单身男人进入,她请他们入座。在已占用的席位之间有 17 个空席位的间隙——在每一端计一个间隙,在相邻的已占席位之间假定是长度为 0 的一个间隙。空席位的总数,即是全部 17 个间隙之和,为 6。假设这些间隙的 x 个有奇数的长度;则 $6 - x$ 个空间可利用来坐各对。(注意 $6 - x$ 为偶且大于等于 0。)现在从左到右,顾客 1,3,5,7,9,11,13,15 中的每一个,其两边有着一个偶数间隙者,吃完饭并离席而去。每个奇数间隙至多防止这 8 个用餐者之一离开,因此至少 $8 - x$ 人留下。仍然仅仅还有 $6 - x$ 个空位可供成对者入座。但是现在有$(8 - x)/2$ 对成对者进入。

38. 这些论证易于推广;对于 $n \geq 1$, $N(n,2) = \lfloor (3n - 1)/2 \rfloor$。[当女招待员使用首先适合的策略以代替最优者时,Robson 已经证明,席位的必要和充分之个数为$\lfloor (5n - 2)/3 \rfloor$。]

39. 把内存分成大小为 $N(n_1, m)$, $N(n_2, m)$和 $N(2m - 2, m)$的三个独立的区域。为处理对空间的请求,对该区域使用有关的最优策略,把每个块放进第一个所述的容量不被超过的区域。这不能失败,因为如果我们对于 x 个单元的要求不能满足,则我们必然至少已经占有了$(n_1 - x + 1) + (n_2 - x + 1) + (2m - x - 1) > n_1 + n_2 - x$ 个位置。

现在若 $f(n) = N(n, m) + N(2m - 2, m)$,则有半加性定律 $f(n_1 + n_2) \leq f(n_1) + f(n_2)$。因此 $\lim f(n)/n$ 存在。(证明:$f(a + bc) \leq f(a) + bf(c)$;因此对于所有的 c, $\limsup_{n\to\infty} f(n)/n = \max_{0 \leq a < c} \limsup_{b\to\infty} f(a + bc)/(a + bc) \leq f(c)/c$;由此 $\limsup_{n\to\infty} f(n)/n \leq \liminf_{n\to\infty} f(n)/n$。)所以 $\lim N(n,m)/n$ 存在。

[由习题 38 我们得知 $N(2) = 3/2$。对于任何 $m > 2$, $N(m)$的值是未知的。不难证明,对于

仅仅两个块大小 1 和 b，乘法因子是 $2-1/b$；因此 $N(3)\geqslant 1\frac{2}{3}$。Robson 的方法意味着 $N(3)\leqslant 1\frac{11}{12}$，而且 $2\leqslant N(4)\leqslant 2\frac{1}{6}$。]

40. 通过使用下列策略，Robson 已经证明 $N(2^r)\leqslant 1+r$：将第一个可用的以 2^m 的倍数开始的 k 单元块，分配给大小为 k 的每个块，其中 $2^m\leqslant k<2^{m+1}$。

当所有块的大小都被限制在处于集合 $\{b_1,b_2,\cdots,b_n\}$ 之中时，命 $N(\{b_1,b_2,\cdots,b_n\})$ 表示乘法因子，则 $N(n)=N(\{1,2,\cdots,n\})$。Robson 和 S. Krogdahl 已经发现每当对于 $1<i\leqslant n$，b_i 是 b_{i-1} 的倍数时，$N(\{b_1,b_2,\cdots,b_n\})=n-(b_1/b_2+\cdots+b_{n-1}/b_n)$；其实，Robson 已经建立了精确的公式 $N=(2^rm,\{1,2,4,\cdots,2^r\})=2^rm(1+\frac{r}{2})-2^r+1$。因此，特别地，$N(n)\geqslant 1+\frac{1}{2}\lceil \lg n\rceil$。他也推出了上界 $N(n)\leqslant 1.1825\ \ln n+O(1)$，而且他试探性地推测 $N(n)=H_n$。一般地说，如果 $N(\{b_1,b_2,\cdots,b_n\})$ 等于 $n-(b_1/b_2+\cdots+b_{n-1}/b_n)$，则这个推测将成立，但遗憾的是并非这种情况，因为 Robson 已经证明 $N(\{3,4\})\geqslant 1\frac{4}{15}$。（见 *Inf. Proc. Letters* **2**（1973），96～97；*JACM* **21**（1974），491～499。）

41. 考虑维护大小 2^k 的块：对于大小 $1,2,4,\cdots,2^{k+1}$ 的要求，将周期地要求分开大小 2^k 的新块，或者恢复该大小的块。通过对 k 用归纳法我们可以证明，被如此分开的诸块所消耗的总的存储，绝不超过 kn；因为在每次要求分开大小 2^{k+1} 的一个块之后，我们在已分开的 2^k 块中至多正使用 kn 个单元，而且在未分开的块中至多正使用 n 个单元。

这个论证可以被加强，以证明有 a_rn 个单元足矣，其中 $a_0=1$ 且 $a_k=1+a_{k-1}(1-2^{-k})$；我们有

$$\begin{array}{ccccccc} k= & 0 & 1 & 2 & 3 & 4 & 5 \\ a_k= & 1 & 1\frac{1}{2} & 2\frac{1}{8} & 2\frac{55}{64} & 3\frac{697}{1024} & 4\frac{18535}{32768} \end{array}$$

反之，对于 $r\leqslant 5$，可以证明，如果把步骤 R1 和 R2 的机制加以修改，以选择分开最坏可能的可用的 2^j 块，而不是第一个这样的块，则一个伙伴系统有时要求 a_rn 个之多的单元。

Robson 关于 $N(2^r)\leqslant 1+r$ 的证明（见习题 40），容易加以修改来证明，一个这样的"最左"的策略，将绝不需要 $(1+r/2)n$ 个以上的单元来为大小 $1,2,4,\cdots,2^r$ 的块分配空间，因为大小 2^k 的块将不被放置在大于等于 $(1+k/2)n$ 的单元。尽管这个算法似乎非常像伙伴系统，但结果是，没有一个伙伴系统有这么好，即使我们把步骤 R1 和 R2 加以修改以选择分开最好可能的可用的 2^j 块也如此。例如，对于 $n=16$ 和 $r=3$，考虑下列内存的"快照"序列：

```
11111111  11111111  00000000  00000000
10101010  10101010  2-2-2-2-  00000000
11110000  11110000  2-110000  00000000
11111111  11110000  11110000  00000000
10101010  10102-2-  10102-2-  00000000
10001000  10002-00  10002-00  4---4---
10000000  10000000  10000000  4---0000
```

这里 0 表示可用单元，而 k 表示 k 块的开始。在类似的方式下，有一系列的操作，每当 n 是 16 的一个倍数时，它迫使大小为 8 的 $\frac{3}{16}n$ 个块成为 $\frac{1}{8}$ 满，而另外的 $\frac{1}{16}n$ 成为 $\frac{1}{2}$ 满。如果 n 是 128 的

倍数，则大小为 8 的$\frac{9}{128}n$ 个块的一个随后的要求，将需要 2.5n 个以上的内存单元。（伙伴系统允许不要的诸 1 溜进 8 块中的$\frac{3}{16}n$ 个，因为在决定性的时间里没有其它可用的诸 2 被分开；"最左"的算法保持所有诸 1 受限制。）

42. 我们可以假定 $m\geq 6$。主要思路是对于 $k=0,1,\cdots$，在内存的开始处建立占用模式 $R_{m-2}(F_{m-3}R_1)^k$，其中 R_j 与 F_j 表示保留的和自由的大小为 j 的块。从 k 到 $k+1$ 的转换以下列方式开始：

$$\begin{aligned}R_{m-2}(F_{m-3}R_1)^k \to\ & R_{m-2}(F_{m-3}R_1)^kR_{m-2}R_{m-2}\to\\ & R_{m-2}(F_{m-3}R_1)^{k-1}F_{2m-4}R_{m-2}\to\\ & R_{m-2}(F_{m-3}R_1)^{k-1}R_mR_{m-5}R_1R_{m-2}\to\\ & R_{m-2}(F_{m-3}R_1)^{k-1}F_mR_{m-5}R_1\end{aligned}$$

然后使用交换序列 $F_{m-3}R_1F_mR_{m-5}R_1\to F_{m-3}R_1R_{m-2}R_2R_{m-5}R_1\to F_{2m-4}R_2R_{m-5}R_1\to R_mR_{m-5}R_1R_2R_{m-5}R_1\to F_mR_{m-5}R_1F_{m-3}R_1$ 共 k 次，直到我们得到 $F_mR_{m-5}R_1(F_{m-3}R_1)^k\to F_{2m-5}R_1(F_{m-3}R_1)^k\to R_{m-2}(F_{m-3}R_1)^{k+1}$为止。最后，当 k 达到足够大时，有一个迫使溢出的"残局"，除非内存容量至少有$(n-4m+11)(m-2)$；细节见于*Comp. J.* **20**（1977），242～244。［注意可想像的最坏情况，它以模式 $F_{m-1}R_1F_{m-1}R_1F_{m-1}R_1\cdots$开始，只不过比这更坏一些而已；习题 6 的下一个适合策略能产生这个悲观的模式。］

43. 我们将证明，如果 $D_1,D_2,\cdots$是对于所有 $m\geq 1$，使得 $D_1/m+D_2/(m+1)+\cdots+D_m/(2m-1)\geq 1$ 的任何数的序列，而且如果 $C_m=D_1/1+D_2/2+\cdots+D_m/m$，则 $N_{\rm FF}(n,m)\leq nC_m$。特别是，由于

$$\frac{1}{m}+\frac{1}{m+1}+\cdots+\frac{1}{2m+1}=1-\frac{1}{2}+\cdots+\frac{1}{2m-3}-\frac{1}{2m-2}+\frac{1}{2m-1}>\ln 2$$

常数序列 $D_m=1/\ln 2$ 满足必要条件。证明通过对 m 用归纳法进行。对于 $j\geq 1$ 令 $N_j=nC_j$，并且假设对于大小为 m 的一个块的某个请求不能在内存最左的 N_m 个单元进行分配。于是 $m>1$。对于 $0\leq j<m$，我们令 N'_j 表示分配给大小小于等于 j 的块最右的位置，或者为 0，如果所有保留的块都大于 j。由归纳法，我们有 $N'_j\leq N_j$。其次我们令 N'_m 是小于等于 N_m 最右被占用的位置，使得 $N'_m\geq N_m-m+1$。于是区间$(N'_{j-1},N'_j]$至少包含$\lceil j(N'_j-N'_{j-1})/(m+j-1)\rceil$个占用的单元，因为它的自由块的大小小于 m 而它的保留块有大小大于等于 j。由此得出 $n-m\geq$已占用单元的个数$\geq\sum_{j=1}^{m}j(N'_j-N'_{j-1})/(m+j-1)=mN'_m/(2m-1)-(m-1)\sum_{j=1}^{m-1}N'_j/(m+j)(m+j-1)>mN_m/(2m-1)-m-(m-1)\sum_{j=1}^{m-1}N_j(1/(m+j-1)-1/(m+j))=\sum_{j=1}^{m}nD_j/(m+j-1)-m\geq n-m$，矛盾。

［这个证明确定了比我们所要求的稍多的结果。如果我们以 $D_1/m+\cdots+D_m/(2m-1)=1$ 来定义诸 D，则序列 $C_1,C_2,\cdots$是 1，7/4，161/72，7483/2880，…；而且如同习题 38 中那样，即使在 $m=2$ 的情况下，这个结果仍可进一步改进。］

44. $\lceil F^{-1}(1/N)\rceil$，$\lceil F^{-1}(2/N)\rceil$，…，$\lceil F^{-1}(N/N)\rceil$。

附录 A　数值数量表

表　1

在标准的子程序中和在计算机程序的分析中，经常使用的数量（到 40 位小数）

$\sqrt{2}=1.41421$	35623	73095	04880	16887	24209	69807	85697 −
$\sqrt{3}=1.73205$	08075	68877	29352	74463	41505	87236	69428 +
$\sqrt{5}=2.23606$	79774	99789	69640	91736	68731	27623	54406 +
$\sqrt{10}=3.16227$	76601	68379	33199	88935	44432	71853	37196 −
$\sqrt[3]{2}=1.25992$	10498	94873	16476	72106	07278	22835	05703 −
$\sqrt[3]{3}=1.44224$	95703	07408	38232	16383	10780	10958	83919 −
$\sqrt[4]{2}=1.18920$	71150	02721	06671	74999	70560	47591	52930 −
$\ln 2=0.69314$	71805	59945	30941	72321	21458	17656	80755 +
$\ln 3=1.09861$	22886	68100	69139	52452	36922	52570	46475 −
$\ln 10=2.30258$	50929	94045	68401	79914	54684	36420	76011 +
$1/\ln 2=1.44269$	50408	88963	40735	99246	81001	89213	74266 +
$1/\ln 10=0.43429$	44819	03251	82765	11289	18916	60508	22944 −
$\pi=3.14159$	26535	89793	23846	26433	83279	50288	41972 −
$1^\circ=\pi/180=0.01745$	32925	19943	29576	92369	07684	88612	71344 +
$1/\pi=0.31830$	98861	83790	67153	77675	26475	02872	40689 +
$\pi^2=9.86960$	44010	89358	61883	44909	99876	15113	53137 −
$\sqrt{\pi}=\Gamma(1/2)=1.77245$	38509	05516	02729	81674	83341	14518	27975 +
$\Gamma(1/3)=2.67893$	85347	07747	63365	56929	40974	67764	41287 −
$\Gamma(2/3)=1.35411$	79394	26400	41694	52880	28154	51378	55193 +
$e=2.71828$	18284	59045	23536	02874	71352	66249	77572 +
$1/e=0.36787$	94411	71442	32159	55237	70161	46086	74458 +
$e^2=7.38905$	60989	30650	22723	04274	60575	00781	31803 +
$\gamma=0.57721$	56649	01532	86060	65120	90082	40243	10422 −
$\ln \pi=1.14472$	98858	49400	17414	34273	51353	05871	16473 −
$\phi=1.61803$	39887	49894	84820	45868	34365	63811	77203 +
$e^\gamma=1.78107$	24179	90197	98523	65041	03107	17954	91696 +
$e^{\pi/4}=2.19328$	00507	38105	45655	97696	59278	73822	34616 +
$\sin 1=0.84147$	09848	07896	50665	25023	21630	29899	96226 −
$\cos 1=0.54030$	23058	68139	71740	09366	07442	97660	37323 +
$-\zeta'(2)=0.93759$	82543	15843	75370	25740	94567	86497	78979 −
$\zeta(3)=1.20205$	69031	59594	28539	97381	61511	44999	07650 −
$\ln \phi=0.48121$	18250	59603	44749	77589	13424	36842	31352 −
$1/\ln \phi=2.07808$	69212	35027	53760	13226	06117	79576	77422 −
$-\ln\ln 2=0.36651$	29205	81664	32701	24391	58232	66946	94543 −

表 2

在标准的子程序中和在计算机程序的分析中经常用到的数量(45 位八进进数字)(出现于“=”号左边的每个量的名称,以十进记法给出)

0.1 = 0.06314	63146	31463	14631	46314	63146	31463	14631	46315 −
0.01 = 0.00507	53412	17270	24365	60507	53412	17270	24365	60510 −
0.001 = 0.00040	61115	64570	65176	76355	44264	16254	02030	44672 +
0.0001 = 0.00003	21556	13530	70414	54512	75170	33021	15002	35223 −
0.00001 = 0.00000	24761	32610	70664	36041	06077	17401	56063	34417 −
0.000001 = 0.00000	02061	57364	05536	66151	55323	07746	44470	26033 +
0.0000001 = 0.00000	00153	27745	15274	53644	12741	72312	20354	02151 +
0.00000001 = 0.00000	00012	57143	56106	04303	47374	77341	01512	63327 +
0.000000001 = 0.00000	00001	04560	27640	46655	12262	71426	40124	21742 +
0.0000000001 = 0.00000	00000	06676	33766	35367	55653	37265	34642	01627 −
$\sqrt{2}$ = 1.32404	74631	77167	46220	42627	66115	46725	12575	17435 +
$\sqrt{3}$ = 1.56663	65641	30231	25163	54453	50265	60361	34073	42223 −
$\sqrt{5}$ = 2.17067	36334	57722	47602	57471	63003	00563	55620	32021 −
$\sqrt{10}$ = 3.12305	40726	64555	22444	02242	57101	41466	33775	22532 +
$\sqrt[3]{2}$ = 1.20505	05746	15345	05342	10756	65334	25574	22415	03024 +
$\sqrt[3]{3}$ = 1.34233	50444	22175	73134	67363	76133	05334	31147	60121 −
$\sqrt[4]{2}$ = 1.14067	74050	61556	12455	72152	64430	60271	02755	73136 +
ln 2 = 0.54271	02775	75071	73632	57117	07316	30007	71366	53640 +
ln 3 = 1.06237	24752	55006	05227	32440	63065	25012	35574	55337 +
ln 10 = 2.23273	06735	52524	25405	56512	66542	56026	46050	50705 +
1/ln 2 = 1.34252	16624	53405	77027	35750	37766	40644	35175	04353 +
1/ln 10 = 0.33626	75425	11562	41614	52325	33525	27655	14756	06220 −
π = 3.11037	55242	10264	30215	14230	63050	56006	70163	21122 +
1° = π/180 = 0.01073	72152	11224	72344	25603	54276	63351	22056	11544 +
1/π = 0.24276	30155	62344	20251	23760	47257	50765	15156	70067 −
π^2 = 11.67517	14467	62135	71322	25561	15466	30021	40654	34103 −
$\sqrt{\pi}$ = Γ(1/2) = 1.61337	61106	64736	65247	47035	40510	15273	34470	17762 −
Γ(1/3) = 2.53347	35234	51013	61316	73106	47644	54653	00106	66046 −
Γ(2/3) = 1.26523	57112	14154	74312	54572	37655	60126	23231	02452 +
e = 2.55760	52130	50535	51246	52773	42542	00471	72363	61661 +
1/e = 0.27426	53066	13167	46761	52726	75436	02440	52371	03355 +
e^2 = 7.30714	45615	23355	33460	63507	35040	32664	25356	50217 +
γ = 0.44742	14770	67666	06172	23215	74376	01002	51313	25521 −
ln π = 1.11206	40443	47503	36413	65374	52661	52410	37511	46057 +
φ = 1.47433	57156	27751	23701	27634	71401	40271	66710	15010 +
e^{γ} = 1.61772	13452	61152	65761	22477	36553	53327	17554	21260 +
$e^{\pi/4}$ = 2.14275	31512	16162	52370	35530	11342	53525	44307	02171 −
sin 1 = 0.65665	24436	04414	73042	03067	23644	11612	07474	14505 −
cos 1 = 0.42450	50037	32406	42711	07022	14666	27320	70675	12321

（续）

$-\zeta'(2) = 0.74001$	45144	53253	42362	42107	23350	50074	46100	27707 +
$\zeta(3) = 1.14735$	00023	60014	20470	15613	42561	31715	10177	06614 +
$\ln\phi = 0.36630$	26256	61213	01145	13700	41004	52264	30700	40646 +
$1/\ln\phi = 2.04776$	60111	17144	41512	11436	16575	00355	43630	40651 +
$-\ln\ln 2 = 0.27351$	71233	67265	63650	17401	56637	26334	31455	57005 −

在本书下一版我将给出这些常量的 36 位十六进制值。

表 1 中的若干 40 位的值是由 John W. Wrench, Jr. 在一台式计算器上计算出来的，用于本书的第 1 版。20 世纪 70 年代用于这样的计算的计算机软件证实了他所计算的所有值都是正确的。关于另外的基本常数的 40 位值，参见习题 1.3.3-23 的答案。

表 3

调和数，伯努利数和斐波那契数的值，对于小的 n 值给出

n	H_n	B_n	F_n	n
0	0	1	0	0
1	1	−1/2	1	1
2	3/2	1/6	1	2
3	11/6	0	2	3
4	25/12	−1/30	3	4
5	137/60	0	5	5
6	49/20	1/42	8	6
7	363/140	0	13	7
8	761/280	−1/30	21	8
9	7129/2520	0	34	9
10	7381/2520	5/66	55	10
11	83711/27720	0	89	11
12	86021/27720	−691/2730	144	12
13	1145993/360360	0	233	13
14	1171733/360360	7/6	377	14
15	1195757/360360	0	610	15
16	2436559/720720	−3617/510	987	16
17	42142223/12252240	0	1597	17
18	14274301/4084080	43867/798	2584	18
19	275295799/77597520	0	4181	19
20	55835135/15519504	−174611/330	6765	20
21	18858053/51731168	0	10946	21
22	19093197/5173168	854513/138	17711	22
23	444316699/118982864	0	28657	23
24	1347822955/356948592	−236364091/2730	46368	24
25	34052522467/8923714800	0	75025	25
26	34395742267/8923714800	8553103/6	121393	26
27	312536252003/80313433200	0	196418	27
28	315404588903/80313433200	−23749461029/870	317811	28
29	9227046511387/2329089562800	0	514229	29
30	9304682830147/2329089562800	8615841276005/14322	832040	30

对于任何 x，命 $H_x = \sum_{n\geqslant 1}\left(\frac{1}{n} - \frac{1}{n+x}\right)$。于是

$H_{1/2} = 2 - 2\ \ln\ 2$

$H_{1/3} = 3 - \frac{1}{2}\pi/\sqrt{3} - \frac{3}{2}\ \ln\ 3$

$H_{2/3} = \frac{3}{2} + \frac{1}{2}\pi/\sqrt{3} - \frac{3}{2}\ \ln\ 3$

$H_{1/4} = 4 - \frac{1}{2}\pi - 3\ \ln\ 2$

$H_{3/4} = \frac{4}{3} + \frac{1}{2}\pi - 3\ \ln\ 2$

$H_{1/5} = 5 - \frac{1}{2}\pi\phi^{3/2}5^{-1/4} - \frac{5}{4}\ \ln\ 5 - \frac{1}{2}\sqrt{5}\ \ln\phi$

$H_{2/5} = \frac{5}{2} - \frac{1}{2}\pi\phi^{-3/2}5^{-1/4} - \frac{5}{4}\ \ln\ 5 + \frac{1}{2}\sqrt{5}\ \ln\phi$

$H_{3/5} = \frac{5}{3} + \frac{1}{2}\pi\phi^{-3/2}5^{-1/4} - \frac{5}{4}\ln\ 5 + \frac{1}{2}\sqrt{5}\ln\phi$

$H_{4/5} = \frac{5}{4} + \frac{1}{2}\pi\phi^{3/2}5^{-1/4}\ \ln\ 5 - \frac{1}{2}\sqrt{5}\ln\phi$

$H_{1/6} = 6 - \frac{1}{2}\pi\sqrt{3} - 2\ \ln\ 2 - \frac{3}{2}\ln\ 3$

$H_{5/6} = \frac{6}{5} + \frac{1}{2}\pi\sqrt{3} - 2\ \ln\ 2 - \frac{3}{2}\ln\ 3$

而且一般地，当 $0 < p < q$（参见习题 1.2.9-19），

$$H_{p/q} = \frac{q}{p} - \frac{\pi}{2}\cot\frac{p}{q}\pi - \ln 2q + 2\sum_{1\leqslant n<q/2}\cos\frac{2pn}{q}\pi\cdot\ln\sin\frac{n}{q}\pi$$

附录B 记号索引

在下列公式中,未作进一步说明的字母的意义如下:

j, k 整数值的算术表达式

m, n 非负整数值的算术表达式

x, y 实数值的算术表达式

f 实数值或复数值的函数

P 指针值的表达式(或是 Λ 或是在一台计算机内的一个地址)

S, T 集合或多重集合

α 符号串

形式符号	意　　义	定义处
$V \leftarrow E$	把表达式 E 的值赋给变量 V	1.1
$U \leftrightarrow V$	交换变量 U 和 V 的值	1.1
A_n 或 $A[n]$	线性数组 A 的第 n 个元素	1.1
A_{mn} 或 $A[m,n]$	矩阵数组 A 的行 m 列 n 中的元素	1.1
NODE(P)	地址为 P(假定 P≠Λ)的节点(由它们的字段名称独立地识别的变量的组)	2.1
F(P)	字段名称是 F 的 NODE(P)中的变量	2.1
CONTENTS(P)	地址是 P 的计算机字的内容	2.1
LOC(V)	在一台计算机内变量 V 的地址	2.1
P⇐AVAIL	把指针变量 P 的值置为一个新节点的地址	2.2.3
AVAIL⇐P	NODE(P)恢复成自由存储;所有它的字段都失去它们的身份	2.2.3
top(S)	在一非空的栈 S 顶上的节点	2.2.1
X⇐S	从 S 弹出到 X:置 X←top(S);然后从非空栈 S 中删去 top(S)	2.2.1
S⇐X	把 X 压入 S:插入值 X 作为栈 S 顶上的一个新的条目	2.2.1
$(B \Rightarrow E; E')$	条件表达式:表示 E 如果 B 为真,表示 E' 如果 B 为假条件	
$[B]$	条件 B 的特征函数:$(B \Rightarrow 1; 0)$	1.2.3
δ_{kj}	克罗内克 δ:$[j = k]$	1.2.3
$[z^n]g(z)$	幂级数 $g(z)$中 z^n 的系数	1.2.9
$\sum_{R(k)} f(k)$	使得 k 为整数且关系 $R(k)$为真的所有 $f(k)$之和	1.2.3
$\prod_{R(k)} f(k)$	使得 k 为整数且关系 $R(k)$为真的所有 $f(k)$之乘积	1.2.3
$\min_{R(k)} f(k)$	使得 k 为整数且关系 $R(k)$为真的所有 $f(k)$之极小值	1.2.3
$\max_{R(k)} f(x)$	使得 k 为整数且关系 $R(k)$为真的所有 $f(k)$之极大值	1.2.3

（续）

形式符号	意　　义	定义处
$j \backslash k$	j 整除 k：$k \bmod j = 0$ 且 $j > 0$	1.2.4
$S \backslash T$	集合差：$\{a \mid a$ 在 S 中但不在 T 中$\}$	
$\gcd(j,k)$	j 与 k 的最大公因子：$(j = k = 0 \Rightarrow 0;\ \max_{d \backslash j, d \backslash k} d)$	1.1
$j \perp k$	j 与 k 互素：$\gcd(j,k) = 1$	1.2.4
A^{T}	矩形数组 A 的转置：$A^{T}[j,k] = A[k,j]$	1.2.3
α^{R}	α 的左右反转	
x^{y}	x 的 y 次方(当 x 为正时)	1.2.2
x^{k}	x 的 k 次方：$(k \geq 0 \Rightarrow \prod_{0 \leq j < k} x;\ 1/x^{-k})$	1.2.2
$x^{\overline{k}}$	k 次递增自乘：$\Gamma(x+k)/\Gamma(x) = (k \geq 0 \Rightarrow \prod_{0 \leq j < k}(x+j);\ 1/(x+k)^{\overline{-k}})$	1.2.5
$x^{\underline{k}}$	k 次递减自乘：$x!/(x-k)! = (k \geq 0 \Rightarrow \prod_{0 \leq j < k}(x-j);\ 1/(x-k)^{\underline{-k}})$	1.2.5
$n!$	n 的阶乘：$\Gamma(n+1) = n^{\underline{n}}$	1.2.5
$\binom{x}{k}$	二项式系数：$(k < 0 \Rightarrow 0;\ x^{\underline{k}}/k!)$	1.2.6
$\binom{n}{n_1, n_2, \cdots, n_m}$	多项式系数(仅当 $n = n_1 + n_2 + \cdots + n_m$ 时才有定义)	1.2.6
$\left[n \atop m \right]$	第一类斯特林数：$\sum_{0 < k_1 < k_2 < \cdots < k_{n-m} < n} k_1 k_2 \cdots k_{n-m}$	1.2.6
$\left\{ n \atop m \right\}$	第二类斯特林数：$\sum_{0 \leq k_1 \leq k_2 \leq \cdots \leq k_{n-m} \leq m} k_1 k_2 \cdots k_{n-m}$	1.2.6
$\{a \mid R(a)\}$	使得关系 $R(a)$ 为真的所有 a 的集合	
$\{a_1, \cdots, a_n\}$	集合或多重集合 $\{a_k \mid 1 \leq k \leq n\}$	
$\{x\}$	小数部分(用于蕴涵的是一个实数值而非集合的范畴内)：$x - \lfloor x \rfloor$	1.2.11.2
$a_1 + a_2 + \cdots + a_n$	n 次和：$\sum_{j=1}^{n} a_j$	1.2.3
$[a,b]$	闭区间：$\{x \mid a \leq x \leq b\}$	1.2.2
(a,b)	开区间：$\{x \mid a < x < b\}$	1.2.2
$[a,b)$	半开区间：$\{x \mid a \leq x < b\}$	1.2.2
$(a,b]$	半闭区间：$\{x \mid a < x \leq b\}$	1.2.2
$\lvert S \rvert$	基数：S 中元数之个数	
$\lvert x \rvert$	x 的绝对值：$(x \geq 0 \Rightarrow x;\ -x)$	
$\lvert \alpha \rvert$	α 的长度	
$\lfloor x \rfloor$	x 的底限，最大整数函数或底限函数：$\max_{k \leq x} k$	1.2.4
$\lceil x \rceil$	x 的顶限，最小整数函数或顶限函数：$\min_{k \geq x} k$	1.2.4
$x \bmod y$	mod 函数：$(y = 0 \Rightarrow x;\ x - y\lfloor x/y \rfloor)$	1.2.4
$x \equiv x' \pmod{y}$	同余关系：$x \bmod y = x' \bmod y$	1.2.4
$O(f(n))$	当 $n \to \infty$ 时，$f(n)$ 的大 O	1.2.11.1

（续）

形式符号	意 义	定义处
$O(f(z))$	当 $z\to 0$ 时，$f(z)$ 的大 O	1.2.11.1
$\Omega(f(n))$	当 $n\to\infty$ 时，$f(n)$ 的 Ω	1.2.11.1
$\Theta(f(n))$	当 $n\to\infty$ 时，$f(n)$ 的 Θ	1.2.11.1
$\log_b x$	以 b 为底，x 的对数（当 $x>0,b>0$ 和 $b\neq 1$ 时）：使得 $x=b^y$ 的 y	1.2.2
$\ln x$	自然对数：$\log_e x$	1.2.2
$\lg x$	x 的二进对数：$\log_2 x$	1.2.2
$\exp x$	x 的指数：e^x	1.2.9
$\langle X_n\rangle$	无穷序列 $X_0,X_1,X_2,\cdots$（这里字母 n 是符号的一部分）	1.2.9
$f'(x)$	f 在 x 处的导数	1.2.9
$f''(x)$	f 在 x 处的二次导数	1.2.10
$f^{(n)}(x)$	第 n 次导数：$(n=0\Rightarrow f(x);g'(x))$，其中 $g(x)=f^{(n-1)}(x)$	1.2.11.2
$H_n^{(x)}$	x 阶的调和数 $=\sum_{1\leqslant k\leqslant n}1/k^x$	1.2.7
H_n	调和数：$H_n^{(1)}$	1.2.7
F_n	斐波那契数：$(n\leqslant 1\Rightarrow n;F_{n-1}+F_{n-2})$	1.2.8
B_n	伯努利数：$n![z^n]z/(e^z-1)$	1.2.11.2
$\det(A)$	方阵 A 的行列式	1.2.3
$\operatorname{sign}(x)$	x 的符号：$[x>0]-[x<0]$	
$\zeta(x)$	ζ 函数：$\lim_{n\to\infty}H_n^{(x)}$（当 $x>1$ 时）	1.2.7
$\Gamma(x)$	Γ 函数：$(x-1)!=\gamma(x,\infty)$	1.2.5
$\gamma(x,y)$	不完全的 Γ 函数：$\int_0^y e^{-t}t^{x-1}\,dt$	1.2.11.3
γ	欧拉常数：$\lim_{n\to\infty}(H_n-\ln n)$	1.2.7
e	自然对数的底：$\sum_{n\geqslant 0}1/n!$	1.2.2
π	圆周率：$4\sum_{n\geqslant 0}(-1)^n/(2n+1)$	
∞	无穷：大于任何数	
Λ	空的链接（不指向地址的指针）	2.1
ϵ	空串（长度为0的串）	
$\emptyset$	空集（无元素的集合）	
ϕ	黄金比，$\frac{1}{2}(1+\sqrt{5})$	1.2.8
$\varphi(n)$	欧拉 φ 函数：$\sum_{0\leqslant k<n}[k\perp n]$	1.2.4
$x\approx y$	x 近似地等于 y	1.2.5,4.2.2
$\Pr(S(X))$	对于 X 的随机值，命题 $S(X)$ 为真的概率	1.2.10
EX	X 的期望值：$\sum_x x\Pr(X=x)$	1.2.10
$\operatorname{mean}(g)$	由生成函数 g 所表示的概率分布的平均值：$g'(1)$	1.2.10
$\operatorname{var}(g)$	由生成函数 g 所表示的概率分布的方差：$g''(1)+g'(1)-g'(1)^2$	1.2.10

（续）

形式符号	意　　义	定义处
(min x_1, ave x_2, max x_3, dev x_4)	一个随机变量，有极小值 x_1，平均(期望)值 x_2，极大值 x_3，标准差 x_4	1.2.10
P *	在一二叉树或树中 NODE(P)的先根序后继之地址	2.3.1,2.3.2
P $	在一二叉树中 NODE(P)的中根序后继之地址，或树中 NODE(P)的后根序后继之地址	2.3.1,2.3.2
P #	在一二叉树中 NODE(P)的后根序后继之地址	2.3.1
* P	在一二叉树或树中 NODE(P)的先根序前驱之地址	2.3.1,2.3.2
$P	在一二叉树中 NODE(P)的中根序前驱之地址，或树中 NODE(P)的后根序前驱之地址	2.3.1,2.3.2
# P	在一二叉树中 NODE(P)的后根序前驱之地址	2.3.1
▌	算法，程序，或证明的结束	1.1
⊔	一个空格	1.3.1
rA	MIX 的寄存器(累加器)A	1.3.1
rX	MIX 的寄存器(扩充)X	1.3.1
rI1,…,rI6	MIX 的(变址)寄存器 I1,…,I6	1.3.1
rJ	MIX 的(转移)寄存器 J	1.3.1
(L:R)	MIX 字的部分字段，0≤L≤R≤5	1.3.1
OP ADDRESS,I(F)	MIX 指令的记号	1.3.1,1.3.2
u	MIX 中的时间单位	1.3.1
*	MIXAL 中的“自身”	1.3.2
0F,1F,2F,…,9F	在 MIXAL 中“向前”的局部符号	1.3.2
0B,1B,2B,…,9B	在 MIXAL 中“向后”的局部符号	1.3.2
0H,1H,2H,…,9H	在 MIXAL 中“这里”的局部符号	1.3.2

索引与词汇表

Some Men pretend to understand a Book
by scouting thro'the Index:
as if a Traveller should go about to describe a Palace
when he had seen nothing but the Privy.
—JONATHAN SWIFT, *Mechanical Operation of the Spirit* (1704)

当一条索引所指页码包含相关习题时,也请参见该习题的答案以获取更多信息。习题答案未参与索引,除非其中有习题中未包含的话题。

( ),见 Identity permutation
0-2-trees 0-2 树 301
  oriented 有向 0-2 树 375
  0-origin indexing 下标从 0 开始 241,267,284—286,290—291
2-d trees 2 维树 534
π 圆周率 19,588—589
  Wallis's product for π 的沃利斯积 48,111
φ 黄金比 11,17,19,77,79—83,588—589

A-register of MIX MIX 的 A 寄存器 121
Aardenne-Ehrenfest, Tatyana van 阿登尼-埃伦费斯特,塔蒂亚纳·范 354,357
Aarons, Roger M. 艾伦斯,罗杰·M 500
Abel, Niels Henrik 阿贝尔,尼亚斯·亨里克 54,467
  binomial thorem 阿贝尔二项式定理 54,68—71,375
  limit theorem 阿贝尔极限定理 92
Absolute error 绝对误差 111
Absolute value 绝对值 19
Absolutely convergent series 绝对收敛级数 26
ACE computer ACE 计算机 184,219
  Pilot 飞行者 ACE 计算机 220
Adams, Charles William 亚当斯,查尔斯·威廉 220
ADD 加指令 127—128,198
Add to list 加到表中,见 Insertion
Addition of polynomials 多项式加法 260—264,337—340
Address 地址:用来标识内存位置的数
  field of MIXAL line MIXAL 语句的地址字段 140,146—148
  of node 节点地址 223
  portion of MIX instruction MIX 指令的地址部分 123
Address transfer operators of MIX MIX 的地址传送操作符 129,201
Adjacent vertices of a graph 图的相邻顶点 343
Adobe Systems Adobe 系统 192
Agenda 日程,见 Priority queue 272,277,282
Aho, Alfred Vaino 阿霍,阿尔弗雷德·维诺 532
Ahrens, Wilhelm Ernst Martin Georg 阿伦斯,威廉·厄恩斯特·马丁·乔治 156
al-Khwārizmī, Abū 'Abd Allāh Muhammad ibn Mūsā 阿科瓦里茨米,阿布·阿布德·阿拉·穆哈默德·伊本·穆萨 2,76
ALF (alphabetic data) 字母数据 146,147,150
Algebraic formulas 代数公式 297
  differentiation 微分(微商) 86,321—330,429
  manipulation of 代数公式的处理 429—433
  representation as trees 表示为树的代数公式 319,431
  simplification of 代数公式的简化 321,329
ALGOL Language ALGOL 语言 192,219
Algorithm, origin of word 算法,词源 1—2

Algorithms 算法 1—8
analysis of 算法分析,见 Analysis of algorithms
communication of 算法通信 14
effective 能行算法 5,7,8
equivalence between 算法间的等价性 437
form of in this book 算法在本书中的形式 2—4
hardware-oriented 面向硬件的算法 24,239,580
how to read 如何读算法 4,14
proof of 算法的证明 4—6,11—15,304—305,342,396—397,407—408
properties of 算法的性质 4—6,8
random paths in 算法中的随机通路 358—360
set-theoretic definition 算法的集合论定义 6—7
theory of 算法论 6,7
Alhazen 阿耳哈曾,一译海桑,见 Ibn al-Haytham
Allocation of tables 表格分配,见 Dynamic storage allocation, Linked allocation, Representation, Sequential allocation
Alpern, Steven Robert 阿尔佩恩,史蒂文·罗伯特 498
Alphameric character 字母数字字符:一个字母,数字或特殊字符
codes for MIX MIX 的字母数字代码 132,133—134,136—137
AMM: American Mathematical Monthly, published by the Mathematical Association of America since 1894 《美国数学月刊》:美国数学协会机关刊物,创刊于 1894 年
Amortized running time 平摊运行时间 241
Analysis of algorithms 算法分析 Ⅶ,6,93—103,163—166,172,238,240,254,263—265,307—309,314,358—360,417—418,423
Analytical Engine 分析机 1,219
Ancestor, in a tree structure 在一树结构中的祖先 296,330
André, Antoine Désiré 安德烈,安托万·德西利 508
Anticipated input 抢先输入,见 Buffering 208
Antisymmetric relation 反对称关系 247
Apostol, Tom Mike 阿波斯托尔,汤姆·迈克 25
Arabic mathematics 阿拉伯数学 1,156
Araújo, Saulo 阿罗约,索罗 536
Arbogast, Louis Francois Antoine 阿波加斯特,路易斯·弗朗索伊斯·安托万 48,101
Arborescences 分叉,见 Oriented trees 343
Arc digraph 有向边有向图 358
Arc in a directed graph 一有向图中的有向边 351
Archive for History of Exact Sciences 《精确科学的历史档案》
Area of memory 内存区域 409
Arguments of subroutines 子程序的自变量 179,180—181
Arithmetic 算术运算:加、减、乘、除 Ⅶ,Ⅸ
fixed-point 定点算术运算 152
floating-point 浮点算术运算 126,291
operators of MIX MIX 的算术操作符 126—129,198
polynomial 多项式算术运算 259—265,338—340,342
scaled decimal 带比例的十进算术运算 154—155
Arithmetic expressions 算术表达式,见 Algebraic formulas
Arithmetic progression, sum of 算术级数的和 10,28—29,52
Array 数组:通常有一 k 维矩阵结构的表 4,222,284—292
linked 链接的数组 287—292
one-dimensional 一维数组,见 Linear list
represented as a tree 表示成树的数组 299
sequential allocation 数组的顺序分配 153,283—287,289—291
tetrahedral 四面体数组 285—286,290—291
two-dimensional 二维数组,见 Matrix
uninitialized 未初始化的数组 292
Arrows, used to represent links in diagrams 箭头,用来表示图中的链接 224
Assembly language 汇编语言:利用符号和记忆名约定来表示机器语言指令,以图简化机器语言程序构造的一种语言
constrasted with high-level language 与高级语言比较 226,50
for MIX MIX 的汇编语言 139—151
Assembly program 汇编程序 140,148
Assertions, inductive 归纳论断 13—18
Assigning a buffer 指定一个缓冲区 209—214,217—218
Assignment operation(←) 赋值运算 3
Asterisk ("*") in assembly language 在一汇编语言中的星号 141,145,148
Asymmetric relations 反对称关系 247

Asymptotic values 渐近值:表示数值量所趋向的极限行为的函数
  derivation of 渐近值的推导 103—119,231,374—375,496—497,536
Atom (in a List) (在一个列表中的)原子 299,384—388,393
  purpose of 原子的功能 386
Automata theory 自动机理论 220,229,434—435
Automation 自动机:形式地定义的抽象机器,它经常试图用做实际计算机某一方面的模型(复数形式是 Automata) 434—435
AVAIL stack AVAIL 栈:可用空间表 243
Available space list 可用空间表 243—247,252,255,264,276,389—390,409—417
  history 可用空间表的历史 430
  variable-size blocks 可变大小的块 409—428
Average value of a random varlable 一个随机变量的平均值 93—94,99
  from a generating function 一个生成函数的随机变量的平均值 96—99

Babbage, Charles 巴贝奇,查尔斯 1,218
Bachmann, Paul Gustav Heinrich 巴克曼,保罗·古斯塔夫·海因里希 103
Backus, John Warner 巴克斯,约翰·沃纳 220
Bailey, Michael John 贝利,迈克尔·约翰 483
Baker, Henry Givens, Jr. 小贝克,亨利·吉文斯 574
Balanced directed graph 平衡有向图 353—354
Ball, Thomas Jaudon 鲍尔,托马斯·耀唐 349
Ball, Walter William Rouse 鲍尔,沃尔特·威廉·劳斯 156
Ballot problem 抽签问题 508—509
Barnes, Ernest William 巴尼斯,欧内斯特·威廉 477
Barnett, Michael Peter 巴尼特,迈克尔·彼得 433
Barrington, David Arno 巴林顿,戴维·阿诺 498
Barry, David McAlister, subtle reference to 巴里,戴维·麦卡利斯特,稍有引用 XI,259
Barton, David Elliott 巴顿,戴维·埃利奥特 63,508
Base address 基地址 232
Bead 珠子,见 Node 223
Before and after diagrams 之前和之后的图式 246—247,263,266,542
Beigel, Richard 贝格尔,理查德 437
Bell, Eric Temple 贝尔,埃里克·坦普尔 84
Bell Interpretive System 贝尔解释系统 220
Bellman, Richard Ernest 贝尔曼,理查德·欧内斯特 XV
Bendix G20 Bendix G20 计算机 120
Bennett, John Makepeace 贝内特,约翰·梅克皮斯 220
Berger, Robert 伯杰,罗伯特 363
Bergeron, Francois 伯杰龙,弗朗索伊斯 372
Bergman, George Mark 伯格曼,乔治·马克 446,563
Berlekamp, Elwyn Ralph 伯利坎普,埃尔温·拉尔夫 258
Berman, Martin Fredric 伯曼,马丁·弗雷德里克 495
Berners-Lee, Mary Lee née Woods 伯内斯-李,玛丽·李·尼·伍兹 220
Bernoulli, Daniel 伯努利,丹尼尔 77
Bernoulli, Jacques(= Jakob = James) 伯努利,雅克(= 雅各布 = 詹姆斯) 108,111
  numbers B_n 伯努利数 74,88,108—111
  numbers, table 伯努利数表 590
  polynomials 伯努利多项式 40,109—111,473
Bernoulli, Jean(= Johann = John), III 伯努利,琼(= 约翰)三世 464
Bertrand, Joseph Louis Francois, postulate 伯特兰,约瑟夫·路易斯·弗朗索伊斯,假设 482
Berztiss, Alfs Teodors 伯齐提斯,阿尔夫斯·特奥多尔斯 433
Best-fit method of storage allocation 存储分配的"最好适合"方法 410—411,420,425—428
Beta function B(x,y) β(贝塔)函数 69
Bhāskarācārya II 巴斯加拉凯里雅二世 50
Bienaymé Irénée Jules 边奈米,埃里尼·朱利斯 97
Bienstock, Daniel 宾斯托克,丹尼尔 485
Big-Oh notation 大 O 记号 103—107,114
Big-Omega notation 大 Ω 记号 106—107
Big-Theta notation 大 Θ 记号 106
Bigelow, Richard Henry 比奇洛,理查德·亨利 533
Binary computer 二进计算机:主要以二进数系进行运算(进制为 2)的计算机
Binary logarithms 二进对数 21,23
Binary number system 二进数系统 22—23
Binary trees 二叉树 296—297,301,302—319,344,431
  complete 完备二叉树 378,382,534
  copying of 二叉树的复制 312—313,316,329

correspondence to trees and forest 二叉树同树和森林的对应 317—318,328
definition of 二叉树的定义 298
"Dewey" notation for 二叉树的杜威记号 301,317,328,382
enumeration of 二叉树的枚举 366—367,382
equivalent 等价二叉树 311—312
erasing of 二叉树的抹去 316
extended 扩充二叉树 376—382
linked 链接二叉树 302
oriented 有向二叉树 374—375
path length of 二叉树的通路长度 376—382
representation of 二叉树的表示 302,305,310,315—316
right-threaded 右穿线二叉树 310,314—316
sequential 顺序二叉树 378
similar 相似二叉树 310—312
threaded 穿线二叉树 305,314—315,431
traversal of 二叉树的遍历 303—304,306,314—315,430—431
with shared subtrees 有共享子树的二叉树 309,572
Binet, Jacques Phillippe Mari- 比内,雅克·菲利普·玛丽 32,383,444,553
Binomial coefficients 二项式系数 48—71,86
asymptotic values 二项式系数的渐近值 69
bounds on 二项式系数的界 71
combinatorial interpretation 二项式系数的组合解释 48—49,70
defined 确定的二项式系数 49
generalized 推广的二项式系数 62,69,81
generating functions 二项式系数的生成函数 87,91
history 二项式系数的历史 49—50
inequalities involving 有关二项式系数的不等式 71
sums involving 有关二项式系数的和 52—71,74—75,81,92
table of 二项式系数表 50
Binomial distribution 二项分布 97—98
tail 二项分布的尾部 102
Binomial number system 二项数系统,见 Combinatorial number system
Binomial theorem 二项式定理 53—54,87
Abel's generalization 二项式定理的阿贝尔推广 68—70,375
generalizations of 二项式定理的推广 61,67—70,87,375—376,457
Hurwitz's generalization 二项式定理的赫维茨推广 376,457
Binomial tree 二叉式树 XX
Bipartite trees 双向树 375
Bit 位,比特:二进数字,0或1
BIT: Nordisk Tidskrift for Informations-Behandling, an international journal published in Scandinavia since 1961 在斯堪的纳维亚出版的国际杂志,创刊于1961年
Bitwise operations 位运算 415,426,480,525
Blaauw, Gerrit Anne 布劳维,格里特·安妮 430
Blackie 布莱克依[英] 44
Blikle, Andrzej Jacek 布利格尔,安德烈·杰西克 312
Block of external data 外部数据块 132—133
Block of memory 内存块 409
Blocking of records 记录的分块 210,216
Bobrow, Daniel Gureasko 博布罗,丹尼尔·格里斯戈 395,431,432
Boles, David Alan 博尔斯,戴维·艾伦 424
Bolzano, Bernard Placidus Johann Nepomuk 布尔柴诺,伯纳德·普拉希达斯·约翰·尼波穆克 360
Boncompagni, Prince Baldassarre 邦康帕尼,普林斯·巴尔达萨利 76
Boothroyd, John 布思罗伊德,约翰 170
Bootstrapping 引导,自举 139
Borchardt, Carl Wilhelm 博查德,卡尔·威廉 382,553
Bottom of stack 栈底 229
Bottom-up process 由底向上的过程 294,333
Boundary tag method of storage allocation 存储分配的边界标记方法 413—415,425—426,432
Bourne, Charles Percy 伯恩,查尔斯·珀西 487
Bracket notation for coefficiens 系数的括号记号 88
Bracket notation for logical statements 逻辑语句的括号记号,见 Iverson's convention 29
Branch instruction 转移指令:有条件的"转移"指令
Branch node of tree 树的分支节点 293
Breadth-first search 宽度优先查找 333
Brenner, Norman Mitchell 布伦纳,诺曼·米切尔 495
Brent, Richard Peirce 布伦特,理查德·皮尔斯 534
Briggs, Henry 布里格斯,亨利 24

Broline, Duane Marvin 布罗莱恩,杜安·马文 570
Brother, in a tree structure 树结构中的兄弟 296
Brouwer, Luitzen Egbertus Jan 布劳威尔,卢特曾·埃格伯杜斯·简 382
Bruijn, Nicolaas Govert de 布鲁恩,尼古拉斯·戈维特·德 117,118,354,358,359,447,474,514,536
Buddy system for storage allocation 存储分配的伙伴系统 415—417,420—421,425—427,431
Buffering of input-output 输入输出缓冲 152,208—219
  history 输入输出缓冲的历史 221
  swapping 转储 143,153,209—210,216
Bugs: Errors or defects 缺陷,见 Debugging
Burke, John 伯克,约翰 295
Burks, Arthur Walter 伯克斯,阿瑟·沃尔特 340
Burleson, Peter Barrus 伯利森,彼得·巴拉斯 433
Burroughs B220 巴勒斯 B220 计算机 120
Burroughs B5000 巴勒斯 B5000 计算机 432
Busche, Conrad Heinrich Edmund Friedrich 布希,康拉德·海因里希·埃德蒙·弗里德里克 39
Busy waiting 忙等待 208
Byte 字节:数据的基本单位,通常与字母数字字符相关联 121
  in MIX MIX 中的字节 120—121,135
Byte size in MIX MIX 中的字节大小:在一个字节中可以存储的不同值的个数

C language C 语言 223,527
Cache 高速缓冲 500
CACM: Communications of the ACM, a pubication of the Association for Computing Machinery since 1958 《ACM 通信》,(美国)计算机协会的刊物,创刊于 1958 年
Cajori, Florian 卡乔里,弗洛里安 22
Calendar 日历 154
California Institute Technology (Caltech) 加利福尼亚理工学院(简称 Caltech) X,267
Call 调用:在一程序中激活另一个子程序
Calling sequence 调用序列 179—182,184,187—188
Campbell, John Arthur 坎贝尔,约翰·阿瑟 422
Canonical cycle notation for permutation 排列的典型循环记法 171—172
Canonical form 范式,典型形式 172
Canonical representation of oriented trees 有向树的典型表示 368—372,374—375,560—561
Capelli, Alfredo 卡佩利,阿尔弗雷多 46,68
Car LISP 语言的术语,用于表示一个列表的头一个分量,类似于 387 页的 INFO 和 DLINK 或 390 页的 ALINK
Cards, playing 扑克牌 47,66,223—226,356—357
Cards punched 穿孔卡片 132—133,147,219
Carlitz, Leonard 卡里兹,伦纳德 92,468,476
Carlyle, Thomas 卡莱尔,托马斯 XIV
Carpenter, Brian Edward 卡彭特,布赖恩·爱德华 216
Carr, John Weber, III 卡尔,约翰·韦伯三世 430
Case Institute of Technology 凯斯技术学院,见原书扉页
Cassini, Gian (= Giovanni = Jean) Domenico (= Dominique 卡西尼,贾恩(=乔瓦尼=琼)·多米尼克 78
Catalan, Eugène Charles 卡塔兰,尤金·查尔斯 383
  numbers 卡塔兰数 383
Cate, Esko George 凯特,埃斯科·乔治 494
Cauchy, Augustin Louis 柯西,奥古斯丁·路易斯 88,444,459,476,492,552
  inequality: $(\sum a_k b_k)^2 \le (\sum a_k^2)(\sum b_k^2)$ 柯西不等式 32
  martrix 柯西矩阵 33—34,444
Cayley, Arthur 凯利,阿瑟 374,382—383,556,566
CDC 1604 CDC 1604 计算机 120,501
Cdr LISP 语言的术语,用来表示删去一个列表的头一个分量之后的剩下部分;类似于 387 页的 RLINK 或 390 页的 BLINK
Ceiling function $\lceil x \rceil$ 顶限函数 35,37
Cell: 单元:计算机存储器的一个字 123
Cellar 窖 229
Central moment of a probability distribution 一个概率分布的中点 100
Centroid of a free tree 一自由树的形心 365—366,374,559
Ceulen, Ludolph van 休伦,卢道夫·范 565
Chain 链:某些作者用来表示一个链接的线性表或一个线性有序集合
Chain rule for diffrentiation 微分的链接规则 48
Chainning 链接:为某些作者用来代替"linking"
Chakravarti, Gurugovinda 查克拉瓦蒂,古鲁戈文达 49

Change of summation variable 求和变量的改变 25,29—30
Channel 通道:连接到一台计算机的数据传输设备 215
CHAR (convert to characters) CHAR(转换成字符)指令 134
Character code of MIX MIX 的字符代码 132,134,136—137
Characteristic function of a probability distribution 概率分布的特征函数 99
Characteristic polynomial of a matrix 矩阵的特征多项式 468
Charles Philip Arthur George of Edinburgh, Prince of Wales 爱丁堡的查尔斯·菲利普·阿瑟·乔治,威尔士王子 395
Cheam, Tat Ong 奇姆,泰特·翁格 422
Cheating 欺骗 552
Chebyshev, Pafnutii Lvovich (Чебышев, Пафнутий Львович) 契比雪夫,帕芙奴蒂·勒沃维茨 96,100
Checkerboard 跳棋盘 409—410
Checkerboarding 跳棋盘化,见 Fragmentation
Chen, Tien Chi 陈天机 440
Cheney, Christopher John 切尼,克里斯托弗·约翰 395
Cheney, Ednah Dow Littlehale 切尼,埃德纳·道·利特黑尔 356
Chernoff, Herman 切诺夫,赫尔曼 472
Chess 国际象棋 5,185,258
Chia Hsien 贾宪 50
Child Link 子链接 402—407
Children in tree structures 树结构中的孩子 296,301,316—317,334
Chinese mathematics 中国数学 50,56,383
Chowla, Paromita 乔拉,帕罗米塔 292
Christian IX, King of Denmark 克里斯琴九世,丹麦国王 295,296,534
Christie Mallowan, Agatha Mary Clarissa Miller 克里斯蒂·马劳安,阿加莎·玛丽·克拉丽莎·米勒 XVI
Chu Shih-Chieh (= Zhū Shì Jié Hànqīng, Zhū Sōngtíng) 朱世杰汉卿,朱松庭 49,55,67
Chung, Fan Rong King 钟金芳蓉 485
Chung, Kai Lai 钟开莱 101
CI MIX 的比较指示器 122,130,135,203,205,218
Circle of buffers 缓冲区圈 215—218,221
Circuit, Eulerian, in a directed graph 在一有向图中的欧拉回路 353—355,358—359,554
Circuit, Hamiltonian, in a directed graph 在一有向图中的哈密顿回路 353,538
Circular definition 循环定义 248,293
Circular linking 循环链接 258—265,287,337,387,431
Circular store 循环存储 229
Circulating shift 循环移位 131
CITRUS 子程序名 429
Clark, Douglas Wells 克拉克,道格拉斯·威尔斯 573
Clavius, Christopher 克拉维尤斯,克里斯托弗 153
Clock, for real time 实时钟 218
Clock, simulated 模拟钟 267,272
Clock, solitaire game 单人游戏钟 356—357
Closed subroutine 闭子程序,见 Subroutine
CMath: Concrete Mathematics 《具体数学》,R. L. Graham、D. E. Knuth 和 O. Patashnik 著 9
CMP1 CMP1 指令(比较 rI1) 130,202—203
CMPA CMPA 指令(比较 rA) 130,202—203
CMPX CMPX 指令(比较 rX) 130,202—203
COBOL: "Common Business-Oriented Language" COBOL 语言:"公共的面向商务的语言" 309—408,429,430
Coding 编码:程序设计的同义词,但用得较少
Coefficient extraction 系数展开 88
Cofactor of element in square matrix 正方矩阵元素的余子式:通过以 1 代替此元素,并以 0 代替和它同行或同列元素而得到矩阵的行列式 33,359
Coffman, Edward Grody, Jr. 小科夫曼,爱德华·格罗迪 422—423
Cohen, Jacques 科恩,雅克 395,433,584
Coin tossing 投掷硬币 97—98
  tail of distribution 硬币分布的尾部 102
Collins, George Edwin 柯林斯,乔治·埃德温 433
Combinations of n objects taken k at a time n 个事物一次取 k 个的组合 48—49,66,91
  with repetitions permitted 允许有重复的 n 个事物一次取 k 个的组合 71,88—89,91,264,366
  with restricted repetitions 有有限制重复的 n 个事物一次取 k 个的组合 91
Combinatorial matrix 组合矩阵 33—34,559
Combinatorial number system 组合数系统 71,532

Comfort Webb T. 康福特,韦布·T 433
COMIT COMIT 语言 433
Command:Synonym for"instruction" 命令:指令的同义词
Comments 注释 2—3
in assembly language 汇编语言中的注释 141,144
Commutative law 交换律 159
Comp. J.: *The Computer Journal*, a publication of the British Computer Society since 1958 《计算机杂志》,英国计算机学会出版的杂志,创刊于1958年
Compacting memory 紧凑内存 397,413,421,424,426
Comparability 可比较性 256
Comparison indicator of MIX MIX 的比较指示器 122,130,135,203,205,218
Comparison operators of MIX MIX 的比较操作符 130,202—203
Compiler 编译程序:翻译计算机语言的程序
algorithms especially for use in 特别在编译程序中用的算法 341,399—400,527
Complete binary tree 完备二叉树 378,381,534
Complete t-ary tree 完备 t 叉树 378—379
Complex conjugate 复数共轭 19
Complex number 复数 19
Compound interest 复利 21—22
Compression of messages 消息压缩 384
Computational error 计算错误 22—24
Computational method 计算方法 4,7—8
Compute 计算:处理数据
Computer 计算机:一台数据处理器
Computer language 计算机语言,见 Assembly language, Machine language, Programming language
CON (constant) CON(常数)字段 144—145,150
Concatenation of strings 字符串的连接 260
Concave function 凹函数 382
Conditional expression 条件表达式 432,592
Congruence 同余 36—38
Connected directed graph 连通有向图 344
strongly 强连通有向图 352,356
Connected graph 连通图 344
Conservative law 守恒定律,见 Kirchhoff's law 164
Constants in assembly language 汇编语言中的常数 144—145,150
Construction of trees 树的构造 322—323,325,403—404
CONTENTS CONTENTS 字段,即"内容"字段 225—226
Context-free grammar 上下文无关文法 510
Continuants 夹行列式,连分数行列式 570
Continued fractions 连分数 468
Continuous simulation 连续模拟 266,285
Convergence 收敛:一无穷序列 $\langle X_n \rangle$ 收敛,如果当 n 趋于无穷时,它趋于一极限;一无穷和或乘积收敛或存在,如果按照微积分的约定,它有值;见等式 1.2.3-(3)
absolute 绝对收敛 26
of power series 幂级数的收敛 84,373
Conversion operators of MIX MIX 的转换操作符 134
Convex function 凸函数 381,566
Convolution of probability distributions 概率分布的卷积:通过把两个独立变量相加得到的分布 99
Conway, Melvin Edward 康伟,梅尔文·爱德华 146,219
Conway, John Horton 康伟,约翰·霍顿 17,77,258,384,570
Copernicus 哥白尼古斯 17
Copy a data structure 一个数据结构的复制,通过产生具有相同数据值和结构关系的另一个不同对象以重复一个有结构的对象
binary tree 复制二叉树 312—313,316,329
linear list 复制线性表 264
List 复制列表 397
two-dimensional linked list 复制二维链接表 292
Copying and compacting 复制与压缩 395
Corless, Robert Malcolm 科利斯,罗伯特·马尔柯姆 372
Coroutines 共行程序 185—191,212—213,267—283,304
history 共行程序的历史 219
linkage 共行程序的链接 186,196,213,276
Correspondence between binary trees and forests 二叉树和森林之间的对应 317—318,328
Cousins 堂兄弟 301
Coxeter, Harold Scott Macdonald 考克斯特,哈罗德·斯科特·麦克唐纳 77,156,383,384
Crain, Steven Paul 克雷恩,史蒂文·保罗 582
Crelle: Journal für die reine und angewandte Mathematik 《纯粹和应用数学杂志》,A.L.Crelle 于1826年

创刊的国际杂志
Crelle, August Leopdd　克雷勒，奥吉斯特·利奥普德　54，602
Critical path time　临界通路时间　208
Crossword puzzle　纵横填字字谜　157
Crowe, Donald Warren　克劳韦，唐纳德·沃伦　570
Cumulants of a probability distribution　概率分布的累加　99—102
Cycle　回路：从顶点到它本身的通路
  detection of　回路的探测　256
  fundamental　基本回路　347—349，356
  in directed graph　有向图中的回路　343
  in graph　图中的回路　343
  in permutation　排列中的回路　158，160，169—171，174—176
  in random permutation　随机排列中的回路　172—176
  notation for permutations　排列的回路记号　158—160，166—169，174
  oriented, in directed graph　在有向图中的有向回路　351
  singleton, in permutation　排列中的单一回路　158，169，172—173

Dahl, Ole-Johan　达尔，奥利-约翰　219，220，432，433
Dahm, David Michael　达姆，戴维·迈克尔　407，408
Data　数据（原来是 datum 一词的复数，但现在用作复数或单数，与“information”相似）：在一种精确的形式化的语言中，对某些事实或概念的表示，通常是可由一计算方法加以处理的数值或字符值　207
  packed　打包的数据　124，152
Data orgainization　数据组织：在一个数据结构中表示数据信息及访问和/或修改该结构的算法的方式
Data structure　数据结构：包括结构关系的一个数据表　222—435
  linear list structures　线性表结构　228—284
  List structures　列表结构　384—398
  multilinked structures　多重链接结构　399—400
  orthogonal lists　正交表结构　285—292，399—408
  tree structures　树结构　293—384
Daughter, in a tree structure　树结构中的女儿　296
David, Florence Nightingale　戴维，弗洛伦斯·奈廷格尔　61
Davies, David Julian Meredith　戴维斯，戴维·朱利安·梅雷迪思　418
Davis, Philip Jacob　戴维斯、菲利普·雅克戴　46
Dawson, Reed　道森，里德　554
de Bruijn, Nicolaas Govert　德布鲁因，尼古拉斯·戈维特　117，118，354，358，359，448，474，514，536
de Moivre, Abraham　棣莫弗，亚伯拉罕　71，79，83，102，175，508
De Morgan, Augustus　德·摩根，奥古斯杜斯　15
Deallocation　释放，见 Liberation
Debugging　排错（调试）：探测和消除错误　184—185，192，243，284，389，527
DEC1（decrease rI1）　DEC1 指令（减 rI1）　130，202
DECA（decrease rA）　DECA 指令（减 rA）　130，202
Decimal computer　十进计算机：主要在十进制数系统中进行运算的计算机
Decimal number system　十进数系统　19，588
DECX（decrease rX）　DECX 指令（减 rX）　130，202
Defined symbol, in assembly language　在汇编语言中确定的符号　148
Definition, circular　循环定义，见 Circular definition
Degree, of node in tree　树中节点的度数　293，301，356
  of vertex in directed graph　有向图中顶点的度数　351
Deletion of node　节点的删除：从一个数据结构消去它，可能把它恢复成可用的存储空间
  from available space list　从可用空间表删除节点，见 Reservation
  from deque　从双端队列删除　238，285
  from doubly linked list　从双链接表删除　266，275—276，285
  from doubly linked ring structure　从双链接环结构删除　339
  from linear list　从线性表删除　229
  from linked list　从链接表删除　225，241，251，290
  from queue　从队列删除　230，232—233，240，247，250，258—259
  from stack　从栈删除　229，230，232—233，235，240，245，255，258—259，263，430
  from tree　从树删除　339
  from two-dimensional list　从二维表删除　290
Demuth, Howard B.　德穆思，霍华德·B　116

Depth-first search 深度优先查找(搜索) 549,551
Deque 双端队列:两个端点的队列 228—231,255
  deletion from 从双端队列删除 239,284
  input-restricted 输入受限双端队列 229—231,391
  insertion into 插入双端队列 238,283
  linked allocation 双端队列的链接分配 265,283
  output-restricted 输出受限双端队列 228—231,255,259
  sequential allocation 双端队列的顺序分配 238
Derangements 扰乱,重排 173,177
Derivative 推导 86,320
Dershowitz, Nachum 德肖维茨,纳丘姆 489,558
Descendant, in a tree structure 在一个树结构中的降序 296
Determinant of a square matrix 正方矩阵的行列式 33—35,78,357—358,360
Deuel, Phillip DeVere, Jr 小德尔,菲利普·德弗里 527
Deutsch, Laurence Peter 多伊奇,劳伦斯·彼得 393,396,397
Dewar, Robert Berriedale Keith 迪尤尔,罗伯特·贝里代尔·基思 583
Dewey, Melvil, notation for binary trees (due to Galton) 杜威,梅尔威尔对于二叉树的记号(归功于高尔顿) 301,313,329,381
  notation for trees 杜威树的记号 297,301,360—361,432
Diaconis, Persi Warren 迪亚科尼斯,珀西·沃伦 460
Diagonals of polygons 多边形的对角 384
Diagrams of structural information 结构信息的图式 224,284
  before-and-after 之前和之后图式 246—247,263,266,542
  binary trees 二叉树的图式 297,302,534—535
  List structures 列表结构图式 299—301,384—385
  tree structures 树结构图式 294—299,319,329,331,431
Dickman, Karl Daniel 迪克曼,卡尔·丹尼尔 497
Dickson, Leonard Eugene 迪克森,伦纳德·尤金 78,453
Dictionaries of English 英语字典 1—2,206—207
Differences of polynomials 多项式的差 64
Differentiation 微分 86,320—329,431
  chain rule for 对微分的链接规则 48
Digamma function $\psi(z)$ 双伽玛函数 40,72,462
Digit 数字:在进制记号中所使用的诸符号之一;通常指十进数字,是符号 0,1,…,9 之一
Digraph 有向图,见 Directed graph 352
Dijkstra, Edsger Wijbe 迪伊克斯特拉,埃德斯盖尔·怀比 15,182,220,221,229,430,433,516,550,574
d'Imperio, Mary Evelyn 德英比里约,玛丽·埃维伦 433
Directed graph 有向图 352—354,397
  as flow chart 作为流程图的有向图 345—346,356
  balanced 平衡有向图 353—354
  connected 连通有向图 344
  regular 正则有向图 358
  strongly connected 强连通有向图 352,356
Discrete system simulation 离散系统的模拟 193,267—285
  synchronous 同步的离散系统模拟 267—285
Disjoint sets 不相交的集合:无公共元素的集合
Disk files 磁盘文件 132—133,409,434
Disk input, buffered 缓冲的磁盘输入 218
Disposal 处理,见 Garbage collection, Liberation
Dissection of a polygon 多边形的剖分 384
Distributed-fit method of storage allocation 存储分配的分布适合方法 422—423,427
Distribution 分布:支配一个随机变量的值的概率的描述
  binomial 二项式分布 97—98
  negative binomial 负二项式分布 103
  normal 正态分布 100—102,118
  Poisson 泊松分布 102,472—496
  tails of 分布的尾部 100,102—103
  uniform 均匀分布,一致分布 98,240,420
Distributive law 分配律 26,33,38,567
  DIV (Divide) DIV (除)指令 127—129,135,199
Divergent series 发散级数 25,73
Division converted to multiplication 除法转换为乘法 488—490
Divisor 因子:如果 $y \bmod x = 0$,则 x 是 y 的一个因式;如果加上 $1 < x < y$,则它是真因子
Dixon, Alfred Cardew 狄克逊,艾尔弗雷德·卡迪尤 459
Dixon, Robert Dan 狄克逊,罗伯特·丹 479
DLINK 向下的链接 385,387

Doran, Robert William 多兰,罗伯特·威廉 216
Double generating function 双重生成函数:两个变量的生成函数 90,313,381,508—510
Double order for traversing trees 遍历树的双重次序 315,316,535
Doubly linked lists 双重链接表 264—265,273—276,284—285,339,387,414,416,424,431
compared to singly linked 与单链接表的比较 265,285
Dougall, John 杜格尔,约翰 459
Dover 多佛 468
Doyle, Arthur Conan 多伊尔,阿瑟·科南 435
Drum memory 磁鼓存储器 132—133,429
Dull, Brutus Cyclops 达尔,布鲁特斯·赛格罗普斯 107
Dummy variable 哑(虚拟)变量 25
Dunlap, James Robert 邓拉普,詹姆斯·罗伯特 429
Dutka, Jacques 杜特卡,雅克 46,61
Dvoretzky, Aryeh 德沃雷兹基,阿耶 563
Dwyer, Barry 德怀尔,巴里 537
Dynamic storage allocation 动态存储分配 234—241,243—245,388—389,409—428
history 动态存储分配的历史 429—430,432—433
running time estimates 动态存储分配运行时间估计 421
Dynastic order 朝代次序,见 Preorder 319
DYSEC computer DYSEC 计算机 221

e 自然对数的底 21,588—589,594
Earley, Jackson Clark 厄尔利,杰克逊·克拉克 433
Easter date 复活节的日期 153—154
Edelman, Paul Henry 埃德尔曼,保罗·亨利 567
Edge in a graph 图的边 344
Edwards, Daniel James 爱德华兹,丹尼尔·詹姆斯 397
Effective algorithm 能行算法 5,7,8
Egorychev, Georgii Petrovich (Епорычев, георгий) 叶戈雷切夫,格奥尔吉·彼得罗维奇 468
Eisele, Peter 艾西尔,彼得 449
Eisenstein, Ferdinand Gotthold Max 艾森斯坦,弗迪南·德·戈德霍尔德·马克斯 448
Elementary symmetric functions 初等对称函数 34,90,466
Elements《几何原本》 2
Elevator (lift) system 电梯系统 267—285
Embedding of partial order into linear order 偏序嵌入到线性序中,见 Topological sorting 248
Embedding of tree in another tree 把树嵌入到另一个树中 330,364
Emulation 模拟,仿真 192
END END 语句 146,150,283
End of file 文件结束 207,218
Endorder 末端次序,见 Postorder
Engles, Robert William 恩格尔斯,罗伯特·威廉 433
English letter frequencies 英文字母的频率 153
ENN1 (enter negative into rI1) ENN1 指令(给 rI1 加负号) 129,202
ENNA (enter negative into rX) ENNA 指令(给 rA 加负号) 129,202
ENNX (enter negative into rX) ENNX 指令(给 rX 加负号) 129,202
ENT1 (enter into rI1) ENT1 指令(给 rI1 赋值) 129,202
ENTA (enter into rA) ENTA 指令(给 rA 赋值) 129,202
Entity 实体,见 Node 223
Entrances to subroutines 子程序入口 178—183
multiple 多个子程序入口 181
ENTX (enter into rX) ENTX 指令(给 rX 赋值) 129,202
Enumeration of subtrees 子树的枚举 356—357
Enumeration of tree structure 树结构的枚举 365—376
history 树结构枚举的历史 382—383
Epictetus of Hierapolis 耶拉波利斯的伊壁鸠鲁 1
EQU (equivalent to) EQU 语句(等价于) 141,144,150
Equivalence algorithm 等价性算法 341—342,548,551
Equivalence classes 等价类 336
Equivalence declaration 等价性说明 341
Equivalence relation 等价关系 335—337,456
Equivalent algorithms 等价算法 437
Equivalent binary trees 等价二叉树 311—312
Equivalent forests 等价森林 328
Equivalent Lists 等价列表 397
Equivalent of a MIXAL symbol 一个 MIXAL 符号的等

价物 150
Equivalent trees 等价树 328
Erase a data structure 抹去一个数据结构:把它的所有节点恢复成可用存储
linear list 抹去线性表 259,260,264
List 抹去列表 388—389
right-threaded binary tree 抹去右穿线二叉树 316
Erdélyi, Arthur 厄尔德莱,阿瑟 376
Erdwinn, Joel Dyne 厄尔德文,乔尔·戴恩 219
Errors, avoiding 避免错误 246—247,527
computational 计算的错误 290
detection of 错误的探测 183—184,191,242,284,389
Etherington, Ivor Malcolm Haddon 埃思林顿,艾弗·马尔科姆·哈登 376
Ettingshausen, Andreas von 埃廷肖欣,安德烈亚斯·冯 50
Etymology 语源,语源学 1—2
Euclides 欧几里得 2,4
algorithm for gcd 求最大公因子的欧几里得算法 2—8,17
algorithm for gcd, extended 扩充的求最大公因子的欧几里得算法 11—12,38
Euclidean domains 欧几里得环 438
Euler, Leonhard(Эйдер, Деонард) 欧拉,伦哈特 45,46,48,53,72,73,84,107,353,383,442,466,508,570
constant γ 欧拉常数 72,103,110,588—589
summation formula 欧拉求和公式 106—111,116,119
theorem 欧拉定理 38
totient function $\varphi(n)$ 欧拉计数函数 $\varphi(n)$ 38,176
Eulerian circuit in directed graph 有向图中的欧拉回路 353—355,358,554
emumeration of 有向图中欧拉回路的枚举 359
Eulerian numbers, second-order 二阶欧拉数 476
Evaluation of powers 幂的计算 479
Evaluation of tree functions 树函数的计算 333,342
Evans, Arthur, Jr 小埃文斯,阿瑟 192
Exchange operation(↔) 交换运算 3,175,259
Exclusive or 异或 415,426,525
Execution time, methods for studying 用于确定执行时间的方法 92,164—165
for MIX instructions MIX 指令的执行时间 134—137
Exercises, notes on 关于习题的说明 268
Exit 出口:控制离开一个子程序的位置
Exits from subroutines 从一些子程序的多个出口 178—183
Expected value of a random varlable 一个随机变量的期望值:平均或"均"值 94,99
from a generating function 一个生成函数的随机变量的期望值 96—99
Exponential generating function for $\langle a_n\rangle$ $\langle a_n\rangle$的指数生成函数:$\sum a_nz^n/n!$ 86
Exponential integral $E_1(x)$ 幂积分 468
Exponents, laws of 指数律 20,23,48
Extended binary tree 扩充的二叉树 376—383
Extended Euclidean algorithm 扩充的欧几里得算法 11—13,38
Extendible, matrix 可扩展矩阵 292
External nodes 外部节点 377—382
External path length 外部通路长度 377,382
Extreme and mean ratio 极大值与均值之比 77

Faà di Bruno, Francesco 伐敌弃,弗朗西斯科 452
Factorials 阶乘 42—48,51
related to gamma function 与伽玛函数有关的阶乘 45
Factorial powers 阶乘的幂 46,48,64,68,95—106
FADD (floating add) FADD 指令(浮点加) 291
Fail-safe program 不安全的程序 256
Fallacious reasoning 不合理推导 16,107,435
Falling powers 降幂 46,64,66,593
Family order 家族序 332,547
sequential representation of trees 树的家族序的顺序表示 332
Family trees 家族树 295—296,301,383
Farber, David Jack 法伯,戴维·杰克 431
Farey, John 法雷,约翰 492
series 法雷级数 155
Father, in a tree structure 在一树结构中的父亲 296
FCMP (floating compare) FCMP 指令(浮点比较) 478,531
FDIV (floating divide) FDIV 指令(浮点除) 291
Ferguson, David Elton 弗格森,戴维·埃尔顿 221,316
Fermat, Pierre de 费马,皮埃尔·德 15,436

theorem 费马定理 37
Ferranti Mark I computer 费伦蒂·马克I计算机 16
Feynman, Richard Phillips 范曼, 理查德·菲利普斯 24
Fibonacci, Leonardo, of Pisa 比萨的斐波那契, 伦纳德 76—77, 81
buddy system 斐波那契伙伴系统 426
generating function 斐波那契生成函数 79—80
number system 斐波那契数系 83, 464
numbers F_n 斐波那契数; 斐波那契序列的元素 11, 16, 76—83
numbers, table of 斐波那契数表 590
sequence 斐波那契序列 11, 16, 76—83, 590
strings 斐波那契串 83
trees 斐波那契树 466
Fibonomial coefficients 斐波那契系数 82, 453, 469
Fich, Faith Ellen 菲西, 费思·埃伦 495
Field 字段: 一个数据集合的指定部分, 通常由连续的(相邻的)符号组成
partial, of MIX word MIX字的部分字段 122—124, 135, 138, 197
within a node 一节点内的字段 223—226
within a node, notations for 在一节点内, 字段的记号 225—226, 430
FIFO 先进先出, 见 Queue 229, 431
Fifty percent rule 百分之五十规则 417—418, 420, 421
Filters 过滤器 189
Final vertex of an arc 一条有向边的终点 351
Fine, Nathan Jacob 法因, 内森·雅克 453
First-fit method of storage allocation 存储分配的首先适合方法 410—412, 425—428, 585
First-in, first-out 先进先出, 见 Queue 229, 333, 431, 516
Fischer, Michael John 费希尔, 迈克尔·约翰 335
Fixed element of permutation 排列的固定元素 158, 173—174
Fixed-point arithmetic 定点算术 152
Flag 标志, 见 Sentinel
Flajolet, Philippe Patrick Michel 弗拉约利特, 菲利普·帕特里克·迈克尔 471, 476, 514, 536
Fletcher, William 弗莱彻, 威廉 499
Floating-point arithmetic 浮点算术 127, 291
operators of MIX MIX的浮点运算符 127, 528—531
Floor function ⌊x⌋ 底限函数 35—37
Flow chart 框图(流程图) Ⅷ, 2—3, 14—16, 344—345, 356
Floyd, Robert W. 弗洛伊德, 罗伯特·W Ⅸ, 15, 17, 18, 397, 437, 478
FLPL FLPL语言 432—433
Flye Sainte-Marie, Camille 弗莱伊·圣玛丽, 卡米尔 554
FMUL (floating multiply) FMUL指令(浮点乘) 291
FOCS: Proceedings of the IEEE Symposia on Foundations of Computer Science (1975—), formerly called *The Symposia on Switching Circuit Theory and Logic Design* (1960—1965), *Symposia on Switching and Automata Theory* (1966—1974) 《IEEE计算机科学基础会议论文集》(1975), 前称为《交换电路理论和逻辑设计论文集》(1960—1965)和《交换和自动机理论论文集(1966—1974)
Ford, Donald Floyd 福特, 唐纳德·弗洛伊德 487
Forecasting 预报 215
Forest 森林: 零个或多个树. 见 Trees 294, 385
correspondence to binary trees 与二叉树对照 316—317, 329
enumeration 森林的枚举 367, 563
index notation for 森林的下标记号 298, 299, 301
Formulas algebraic 代数公式, 见 Algebraic formulas
Formulæ de Serierum Reversione 《级数反演公式》
Förstemann, Wilhelm August 福斯特曼, 威廉·奥古斯特 459
FORTRAN language FORTRAN语言 221, 223, 282, 341, 430, 432
Foster, Frederic Gordon 福斯特, 弗雷德里克·戈登 96
Fourier, Jean Baptiste Joseph 傅里叶, 琼·巴普蒂斯特·约瑟夫 25
Fractional part 小数部分 36
Fraenkel, Aviezri S 弗伦克尔, 阿维兹里·S 238
Fragmentation 碎片 413, 422, 423, 428
Fredman, Michael Lawrence 弗雷德曼, 迈克尔·劳伦斯 485
Free lattice 自由格 330
Free storage 自由存储, 见 Available space
Free subtress 自由子树 345—350
enumeration of 自由子树的枚举 357—358

minimum cost 极小代价自由子树 351
Free trees 自由树 345—351
definition of 自由树的定义 343
enumeration of 自由树的枚举 357—366,375,383
Freeman 弗里曼 17
Friedman, Daniel Paul 弗里德曼,丹尼尔·保罗 396
Frieze patterns 中楣模式 384—385
Front of queue 队列的前端 230
FSUB (floating subtract) FSUB 指令(浮点减) 291
Fuchs, David Raymond 富克斯,戴维·雷蒙德 192
Fukuoka, Hirobumi 福冈博文 478
Full word logical (bitwise) operations 全字逻辑(位)运算 415,426,480,525
Fundamental cycles in graph 图中的基本回路 346—350,356
Fundamental path 基本通路 348
Fundamental theorem of arithmetic 算术基本定理 38
Furch, Robert 弗兹,罗伯特 117
Future reference in MIXAL MIXAL 中的未来的访问 148,150
restrictions on 对未来访问的限制 150

Galler, Bernard Aron 加勒,伯纳德·艾伦 335
Galton, Francis 高尔顿,弗朗西斯 534,603
Games, solution of 一些游戏的解 83,257—258
Gamma function Γ(z) 伽玛函数 45—48,69,76,112—115
incomplete 不完备的伽玛函数 113—118
Gao, Zhicheng 高志成 536
Garbage collection 废料收集,无用信息收集 243,389—398,412—413,421,426,432,517,522
efficiency of 废料收集的效率 395—396
Gardner, Martin 加德纳,马丁 17,77,557
Garwick, Jan Vaumund 加威克,简·沃蒙德 236,429
Gaskell, Robert Eugene 加斯克尔,罗伯特·尤金 83
Gasper, George, Jr. 小加斯珀,乔治 459
Gates, William Henry, III 盖茨,威廉·亨利三世 XI
Gauβ (= Gauss), Johann Friedrich Carl (= Carl Friedrich) 高斯,约翰·弗里德里克·卡尔(=卡尔·弗里德里克) 45,54,91
gcd: Greatest common divisor 最大公因子
Gelernter, Herbert Leo 格伦特,赫伯特·利奥 432
Generating functions 生成函数 79—81,83—92,231,364—366,369—370,372—376,510
double 双重生成函数 91,374,382,508—510
for discrete probability distributions 离散概率分布的生成函数 95—103,174
Genuys, Francois 吉奴伊斯,弗朗索伊斯 221
Geometric progression, sum of 几何级数的和 28,85
Gerberich, Carl Luther 格伯里奇,卡尔·卢瑟 432
Gill, Stanley 吉尔,斯坦利 219,220,429
Giornale di Matematiche di Battaglini 《巴达格里尼数学学报》
Girard, Albert 吉拉德,阿尔伯特 466
Glaisher, James Whitbread Lee 格拉西尔,詹姆斯·惠特布雷德·李 474
Glassey, Charles Roger 格拉西,查尔斯·罗杰 382
Glasgow 格拉斯哥[英] 44
Gnedenko, Boris Vladimirovich (Гнеденко, Борис Владимирович 格涅坚科,鲍利斯·弗拉基米罗维奇 101
GO-button of MIX MIX 的 GO-按钮 122,134,139,203
Goldbach, Christian 哥德巴赫,克里斯琴 45,442
Goldberg, Joel 戈德堡,乔尔 500
Golden ratio 黄金比 12,17,19,77,80—83,588—589
Goldman, Alan Joseph 戈德曼,艾伦·约瑟夫 559
Goldstine, Herman Heine 戈德斯坦,赫尔曼·海因 17,219,221
Golomb, Solomon Wolf 戈龙卜,索罗门·沃尔夫 176
Golumbic, Martin Charles 戈卢布克,马丁·查尔斯 566
Goncharov, Vasilii Leonidovich (Гончаров, Васылий Леонидович 贡恰罗夫,瓦西里·利奥尼多维奇 471
Gonnet Haas, Gaston Henry 戈内特·哈斯,盖斯顿·亨利 373
Good, Irving John 古德,欧文·约翰 353,373,451,554
Gopāla 戈帕拉 77
Gorn, Saul 戈恩,索尔 432
Gosper, Ralph William, Jr. 小戈斯珀,拉尔夫·威廉 62
Gould, Henry Wadsworth 古尔德,亨利·沃兹沃思 54,60,117,453,461
Gourdon, Xavier Richard 古尔登,泽维尔·理查德 497
Gower, John Clifford 高尔,约翰·克利福德 431
Grabner, Peter Johannes 格拉布内,彼得·约翰尼斯

476
Graham, Ronald Lewis 格雷厄姆,罗纳德·刘易斯 9
Graphs 图 343—352,434
directed 有向图,见 Directed graph
Greatest common divisor 最大公因子 2—8,11—12,36,78—79
Greatest integer function 最大整数函数,见 Floor function
Greek mathematics 希腊数学 2,17,50,563
Griswold, Ralph Edward 格里斯沃尔德,拉尔夫·爱德华 432
Grounded wire symbol 接地符号 224
Grünbaum, Branko 格伦鲍姆,布兰科 362,557
Gustavson, Fred Gehrung 古斯塔弗森,弗雷德·格伦 290
Guy, Richard Kenneth 盖伊,理查德·肯尼思 18,77,258,570

H-trees H 树 535
Haddon, Bruce Kenneth 哈登,布鲁斯·肯尼思 583
Hadeler, Karl-Peter Fritz 哈德勒,卡尔-彼得·弗里茨 449
Hageman, Louis Alfred 哈格曼,路易斯·艾尔弗雷德 556
Halāyudha 哈拉尤达 49
Hamel, Georg 哈默尔,乔治 449
Hamilton, William Rowan, circuit 哈密顿,威廉·罗恩线路 353,357
Hamlet, Prince of Denmark 哈姆赖特,丹麦王子 222
Hamming, Richard Wesley 汉明,理查德·韦斯利 24
Hankel, Hermann 汉克尔,赫尔曼 45
Hansen, James Rone 汉森,詹姆斯·罗恩 432
Hansen, Wilfred James 汉森,威尔弗雷德·詹姆斯 395
Haralambous, Yannis 哈拉拉姆博斯,耶尼斯 630
Harary, Frank 哈拉里,弗兰克 383
Hardware-oriented algorithms 面向硬件的算法 24,239,580
Hardy, Godfrey Harold 哈迪,戈弗雷·哈罗德 12,382,461,492
Hare, David Edwin George 黑尔,戴维·埃德温·乔治 373
Hare and hounds 兔子与猎犬,见 Military game
Harmonic numbers H_n 调和数 72—76,110
generating function 调和数生成函数 87
table 调和数表 590—591
Harmonic series 调和级数 72,154—155
Haros, C. 哈罗斯·C 492
Hartmanis, Juris 哈特曼尼斯,朱里斯 434
Harvard University Press 哈佛大学出版社
Hautus, Matheus Lodewijk Johannes 豪道斯,马瑟斯·洛德威克·约翰尼斯 458
hcf 见 gcd
Head of list 表头,见 List head
Heap 堆,见 Pool of available nodes 409
Height of tree or forest 树或森林的高度 536
Heine, Heinrich Eduard 海因,海因里希·埃杜瓦德 459
Hellerman, Herbert 赫勒曼,赫伯特 431
Hemacandra, Ācārya 赫马坎德拉,阿卡里亚 78
Henkin, Léon Albert 亨金,利昂·艾伯特 15
Henrici, Peter Karl Eugen 亨里西,彼得·卡尔·尤金 84
Herbert, George 赫伯特,乔治 XIII
Hermite, Charles 赫密特,查尔斯 45,447
Hesse-Kassel, Louise Wichelmine Friederike Karoline Auguste Julia von 赫西-卡塞尔,路易斯·威廉明·弗里德里克·卡罗林·奥古斯特·朱丽娅·冯 295,296
Heyting, Arend 海汀,阿伦 382
Hilbert, David, matrix 希尔伯特,戴维矩阵 34
Hiles, John Owen 海尔斯,约翰·欧文 395
Hindu mathematics 印度数学 50,76
Hill, Robert 希尔,罗伯特 490
Hipparchus of Nicæa 尼卡的希帕楚斯 563
HLT (halt) HLT 指令(停机) 131,138
Hoare, Charles Antony Richard 霍尔,查尔斯·安东尼·理查德 15,220,432,433
Hobbes, Thomas 霍比斯,托马斯 630
Hobby, John Douglas 霍比,约翰·道格拉斯 630
Hofri, Micha 霍夫里,迈卡 486
Holmes, Thomas Sherlock Scott 霍姆斯,托马斯·舍洛克·斯科特 436
Holt Hopfenberg, Anatol Wolf 霍尔特·霍芬贝格,阿纳托尔·沃尔夫 431
Honeywell H800 霍尼韦尔公司 H800 计算机 120
Hopcroft, John Edward 霍普克罗夫特,约翰·爱德华 531
Hopper, Grace Brewster Murray 霍珀,格雷斯·布鲁斯

特·默里 219
Hu, Te Chiang 胡德强 381,566
Huang Bing-Chao 黄秉超 169
Huffman, David Albert 赫夫曼,戴维·艾伯特 378,383
algorithm 赫夫曼算法 378—382
Hurwitz, Adolf 赫尔维茨,阿道夫 40
binomial theorem 赫尔维茨二项式定理 376,457
Hwang, Frank Kwangming 黄光明 382,565
Hwang, Hsien-Kuei 黄显贵 63
Hyperfactorial 超阶乘 111
Hypergeometric functions 超几何函数 61
basic 基本超几何函数 459

I/O: Input or output 输入/输出 207
I1-register of MIX MIX 的 I1 寄存器 121,135
IBM 650 IBM 公司 650 计算机 Ⅰ,120,220,501
IBM 701 IBM 公司 701 计算机 220
IBM 705 IBM 公司 705 计算机 220
IBM 709 IBM 公司 709 计算机 120,501
IBM 7070 IBM 公司 7070 计算机 120
Ibn al-Haytham, Abū 'Alī al-Hasan 伊本·阿尔-海山,阿卜·阿里·阿尔·哈桑 156
Identity permutation 恒等排列 158,168
Iliffe, John Kenneth 艾利夫,约翰·肯尼思 434
Illiac I computer 伊利阿克 I 计算机 220
Imaginary part of complex number 复数的虚部 21
d'Imperio, Mary Evelyn 德因佩里奥,玛丽·伊夫林 434
IN (input) IN 指令(输入) 132,207—208
In-degree of vertex 顶点的入度 351
in situ permutation 原位排列 8,158,177,495
INC1 (increase rI1) INC1 指令(增 rI1) 129,202
INCA (increase rA) INCA 指令(增 rA) 129,202
Incidence matrix 伴随矩阵 256
Inclusion and exclusion principle 容斥原理 174,176
Incomplete gamma function $\gamma(a,x)$ 不完备的伽玛函数 112—118
INCX (increase rX) INCX 指令(增 rX) 129,202
Indentation 缩进 297
Index 下标:用于指出一数组的具体元素的数(常称为"subscript") 3,485,298,299
Index register 变址寄存器 121,123,152,252
modification of MIX instructions MIX 指令的变址寄存器的修改 123,238—239
Index variable 下标变量 25
Indian mathematics 印度数学 50,76
Indirect addressing 间接编(寻)址 234,238—239,291
Induction, mathematical 数学归纳法 9—18,29,300,444
generalized 广义数学归纳法 17
Inductive assertions 归纳论断 13—17
Inductive closure 归纳终止 444
Infinite series 无穷级数:对于无穷多值的求和 25—27,54,83—92
Infinite trees 无穷树 301,360
Infinity lemma 无穷性引理 360,364
Information 信息:与数据相关联的意义,数据所表示的事实或概念;通常在狭义下也用做"数据"的同义语,或者在更广的意义下用做可由数据导出的任何概念
Information structure 信息结构,见 Data structure
Ingalls, Daniel Henry Holmes 英戈尔斯,丹尼尔·亨利·霍姆斯 494
Intial vertex of an arc 有向边的起点 351
Inorder for a binary tree 二叉树的中根序 303—306,313—314,329
Input 输入 4,206—218
anticipated 抢先输入 153,207,215
buffering 输入缓冲 153,206—218,221
operators of MIX MIX 的输入操作符 132—134,206—207
Input-restricted deque 输入受限双端队列 229—233,391
Insertion of a node 节点的插入:把节点放入一个数据结构中
into available space list 插入可用空间表,见 Liberation
into deque 插入双端队列 238,284
into doubly linked list 插入双重链接表中 266,275,284
into doubly linked ring structure 插入双重链接环结构 339
into linear list 插入线性表中 228
into linked list 插入链接表中 225,241,261,290
into quadruply linked binary tree 插入四重链接二叉树中 316

into queue 插入队列中 230,232—233,240,246,251,259—260
into threaded binary tree 插入穿线二叉树中 310,315
into two-dimensional list 插入二维表中 290
onto a stack 插入到栈上 229,230,232—233,235,240,244,255,259—260,264,429
Instruction, machine language 机器语言指令:一个代码,当由一计算机线路解释它时,引起计算机实施某一动作
in MIX MIX 的指令 123—140
symbolic form 指令的符号形式 124,140
INT (interrupt) INT 指令(中断) 215
Integer 整数 18
Integration 积分 87
by parts 分部积分 74,108—109
related to summation 同求和有关的积分 107—112
Interchange of values(↔) 诸值的互换 3,175,259
Interchanging the order of summation 交换求和的次序 27,30,32,39
Interest, compound 复利 20—21
Interlock time 互锁时间:在一部分忙于完成某个动作时,延迟系统的另一部分
Internal nodes 内部节点 377—382
Internal path length 内部通路长度 377,379,382
Internet 因特网 Ⅱ,Ⅺ,631
Interpreter (interpretive routine) 解释程序 191—193,220,322
Interrupt 中断 218
Intervals, notation for 区间的记号 19
Invariants 不变量 15
Inverse modulo m 模 m 逆 38
Inverse of a matrix 矩阵的逆 33—34,70,292
Inverse of a permutation 排列的逆 102,168—171,175
Inversion problem 反演问题 60—61
Inversions of permutation 排列的反演 513,528,541
Inverting a linked list 颠倒一个链接表 255,264
IOC (input-output control) IOC (输入输出控制) 133
IPL IPL 语言 220,430—431,432—433,523
Irrational radix 无理基数 83
Irreflexive relation 非反身关系 247
Isaacs, Irving Martin 伊萨克斯,欧文·马丁 571
Islamic mathematics 伊斯兰数学 2,156
Isolated vertex 孤立顶点 353
Itai, Alon 伊泰,阿龙 506
Iverson, Kenneth Eugene 艾弗森,肯尼思·尤金 30,35,431—432
convention 艾弗森约定 29—30,58,99

J-register of MIX MIX 的 J 寄存器 121,138,178,181,204—206
J1N (jump if rI1 negative) J1N 指令(若 rI1 负则转移) 131,202
J1NN (jump if rI1 nonnegative) J1NN 指令(若 rI1 非负则转移) 131,202
J1NP (jump if rI1 nonpositive) J1NP 指令(若 rI1 非正则转移) 131,202
J1NZ (jump if rI1 nonzero) J1NZ 指令(若 rI1 为零则转移) 131,202
J1P (jump if rI1 positive) J1P 指令(若 rI1 正则转移) 131,202
J1Z (jumpif rI1 zero) J1Z 指令(若 rI1 为零,则转移) 131,202
JACM: *Journal of the ACM*, a publication of the Association for Computing Machinery since 1954 《ACM 杂志》(美国)计算机协会刊物
Jacob, Simon 雅克,西蒙 78
Jacquard, Joseph Marie 雅克夸德,约瑟夫·玛丽 219
Jacquet, Phillippe Pierre 雅克,菲利普·皮埃尔 486
JAN (jump if rA negative) JAN 指令(rA 为负则转移) 131,202
JANN (jump if rA nonnegative) JANN 指令(rA 非负则转移) 131,202
JANP (jump if rA nonpositive) JANP 指令(rA 非正则转移) 131,202
JANZ (jump if rA nonzero) JANZ 指令(rA 非零则转移) 131,202
JAP (jump if rA positive) JAP 指令(rA 为正则转移) 131,202
Japanese mathematics 日本数学 108,111
Jarden, Dov 贾登,达夫 82,460
Java Language Java 语言 223
JAZ (jump if rA zero) JAZ 指令(rA 为 0 则转移) 131,202
JBUS (jump if busy) JBUS 指令(忙碌时转移)

133,151,204,208
JE (jump if equal) JE 指令(相等时转移) 131,201
Jeffrey, David John 杰弗里,戴维·约翰 372
Jenkins, David Philip 詹金斯,戴维·菲利普 432
JG (jump if greater) JG 指令(大于时转移) 131,201
JGE (jump if greater-or-equal) JGE 指令(大于或等于时转移) 131,201
JL (jump if less) JL 指令(小于时转移) 131,201
JLE (jump if less-or-equal) JLE 指令(小于或等于时转移) 131,201
JMP (jump) JMP 指令(转移) 130,179,201,273
JNE (jump if not equal) JNE 指令(不等时转移) 131,201
JNOV (jump if no overflow) JNOV 指令(无溢出时转移) 130,137,201
Jodeit, Jane Griffin 乔德特,简·格里芬 434
Johnson, Lyle Robert 约翰逊,莱尔·罗伯特 431—432
Johnstone, Mark Stuart 约翰斯通,马克·斯图尔特 424
Jokes 笑话 50,190
Jones, Clifford Bryn 琼斯,克利福德·布里恩 16
Jones, Mary Whitmore 琼斯,玛丽·惠特莫尔 357
Jonkers, Henricus (= Hans) Bernardus Maria 琼克斯,亨里克斯(= 汉斯)·伯纳德斯·玛丽娅 583
Jordan, Marie Ennemond Camille 乔丹,玛丽·恩尼蒙德·卡米尔 365,383
Josephus, Flavius, sun of Matthias, problem 约瑟夫,弗雷维厄斯,马赛厄斯之子,约瑟夫问题 156,176
JOV (jump if overflow) JOV 指令(溢出时转移) 130,137,201
Joyal, André 乔耶尔,安德烈 373
JRED (jump if ready) JRED 指令(准备好则转移) 133,212—213
JSJ (jump saving rJ) JSJ 指令(转移,保存 rJ) 130,180,202,503
Jump operators of MIX MIX 的转移操作符 130—131,201
Jump trace 转移踪迹 206,283,500
JXN (jump if rX negative) JXN 指令(rX 为负则转移) 131,202
JXNN (jump if rX nonnegative) JXNN 指令(rX 非负时转移) 131,202
JXNP (jump if rX nonpositive) JXNP 指令(rX 非正则转移) 131,202
JXNZ (jump if rX nonzero) JXNZ 指令(rX 非 0 则转移) 131,202
JXP (jump if rX positive) JXP 指令(rX 为正则转移) 131,202
JXZ (jump if rX zero) JXZ 指令(rX 为 0 则转移) 131,202

Kahn, Arthur Bertram 卡恩,阿瑟·贝特拉姆 254
Kahrimanian, Harry George 卡里美尼安,哈里·乔治 431
Kallick, Bruce 卡利克,布鲁斯 381
Kaplansky, Irving 卡普兰斯基,欧文 176
Karamata, Jovan 卡拉马塔·乔万 63
Karp, Richard Manning 卡普,理查德·曼宁 383
Katz, Leo 卡茨,利奥 560
Kaucký, Josef 考克基,约瑟夫 60
Keller, Helen Adams 凯勒,海伦·亚当斯 119
Kepler, Johannes 开普勒,约翰尼斯 77,78
Kilmer, Alfred Joyce 基尔默,艾尔弗雷德·乔伊斯 222
King, James Cornelius 金,詹姆斯·科尼利厄斯 18
Kirchhoff, Gustav Robert 基尔霍夫,古斯塔夫·罗伯特 383,553
law of conservation of flow 基尔霍夫流量守恒定律 93,164—165,254,263,344—350,359
Kirkman, Thomas Penyngton 科克曼,托马斯·彭宁顿 385
Kirschenhofer, Peter 基尔申霍夫,彼得 476
Klarner, David Anthony 克拉那,戴维·安东尼 83
Kleitman, Daniel J. (Isaiah Solomon) 克莱特曼,丹尼尔·J(艾赛亚·所罗门) 518,565
Knopp, Konrad Hermann Theodor 诺普,康拉德·赫尔曼·西奥多 44,468
Knotted lists 打结的表 431
Knowlton, Kenneth Charles 诺尔顿,肯尼思·查尔斯 433
Knuth, Donald Ervin 克努特,唐纳德·欧文(高德纳) Ⅱ,Ⅳ,Ⅺ,9,30,63,117,185,191,192,282,283,373,429;333,441,452,469,474,495,497,536,549,550,554,562,566,600,630
Knuth, Nancy Jill Carter 克努特,南茜·吉尔·卡特(高精兰) Ⅹ,ⅩⅩ

Kolmogorov, Andrei Nikolaevich (Колмогоров, Андрей Нниколавнч) 科尔莫戈罗夫,安德烈·尼古拉耶维奇 100,101,435
König, Dénes 科尼格,德尼斯 360,382,558
Koster, Cornelis(=Kees) Hermanus Antonius 科斯特,科尼利斯(=基斯)·赫尔曼纽斯·安东尼亚斯 433
Kozelka, Robert Marvin 科泽尔卡,罗伯特·马文 516
Kramp, Christian 克兰普,克里斯琴 45
Krattenthaler, Christian 克拉腾特勒,克里斯琴 35
Kreweras, Germain 克鲁韦拉斯,杰曼 568
Krogdahl, Stein 克罗格达尔,斯坦恩 585
Kronecker, Leopold, delta notation 克罗内克,利奥波德 δ记号 30,58,592
Kruskal, Joseph Bernard 克鲁斯科尔,约瑟夫·伯纳德 364,558
Kummer, Ernst Eduard 库莫尔,厄恩斯特·埃杜瓦德 67
Kung, Hsiany Tsung 孔祥重 534

Labeled trees, enumeration of 带标号树的枚举 366,383
Labelle, Gilbert 拉贝尔,吉尔伯特 372
Lagrange(=dela Grange), Joseph Louis, Comte 拉格朗日,约瑟夫·路易斯,科姆特
inversion formula 拉格朗日反演公式 369,563
Lamé, Gabriel 拉姆,加布里埃尔 383
Lamport, Leslie B. 兰波特,莱斯利·B 574
Language 语言:符号串的集合,通常伴之以对集合中每个串赋加"意义"的一些约定 5,230,432—433
machine 机器语言 Ⅷ—Ⅹ,120
Laplace(=de la Place), Pierre Simon, Marquis de 拉普拉斯,皮埃尔·西蒙,马奎斯·德 84
transform 拉普拉斯变换 91
Laplacian of graph 图的拉普拉斯矩阵 553
Lapko, Olga Georgievna (Лапко, Олвга Георгиевна) 拉普科,奥尔加·格奥尔吉维娜 630
Large programs, writing 编写大型程序 183—185
Larus, James Richard 拉鲁斯,詹姆斯·理查德 349
Last-in-first-out 后进先出,见 Stack 229,424,431
almost 几乎后进先出 419,422,427
Latency 等待时间 218,429
Lattice 格:产生∩和∪之类操作的代数系统
defined on forests 在森林中定义的格 547,568
free 自由格 330
Lawson, Harold Wilbur, Jr. 小劳森,哈罗德·威尔伯 407,433
LCHILD LCHILD字段 334—335,340—341
LD1 (load rI1) LD1 指令(装入 rI1) 125,199
LD1N(load rI1 negative) LD1N 指令(装入负的 rI1) 125,199
LDA (load A) LDA 指令(装入 rA) 125,199
LDAN (load rA negative) LDAN 指令(装入负的 rA) 125,199
LDX (load rX) LDX 指令(装入 rX) 125,199
LDXN (Load rX negative) LDXN 指令(装入负的 rX) 125,199
Leaf of tree 树的叶,见 Terminal node 293
Least-recently-used replacement 替换最近最少使用的 424
Leeuwen, Jan van 利尤文,简·范 566
Left subtree in a binary tree 二叉树中的左子树 297,302
Left-to-right mnaximum or minimum 自左至右极大值或极小值 93—97,100—102,172
Legendre(=Le Gendre), Adrien Marie 勒让德,艾德里安·玛丽 45,47
symbol 勒让德符号 41
Léger, Émile 莱杰,埃米尔 77
Lehmer, Derrick Henry 莱默,德里克·亨利 435
Leibniz, Gottfried Wilhelm Freiherr von 莱布尼茨,戈特弗里德·威廉·弗雷赫尔·冯 2,47
Leighton, Frank Thomson 莱顿,弗兰克·汤姆逊 422—423
Leiner, Alan Lewine 莱因尼尔,艾伦·卢因 221
Leipzig 莱比锡[德] 2,59,383,456
Leonardo of Pisa 比萨的伦纳德 76—77,81
Leroux, Pierre 勒鲁,皮埃尔 372
Letter frequencies in English 英语中的字母频率 153
Level of node in a tree 树中节点的级 293,300,301
Level-order 层次序 333,535,547
sequential representation 层次序的顺序表示 333,340
LeVeque, William Judson 莱维克,威廉·贾德森 436
Lévy, Paul 利维,保罗 101
Levy, Silvio Vieira Ferreira 利维,西尔维奥·维拉·费雷拉 Ⅺ

Lexicographic order 字典序 18,285—286,291,535
Liberation of reserved storage 保留存储的释放 243,245,277,389—390,395—396,411—414
LIFO 后进先出,见 Stack 229,431
Lilius,Aloysius 莉留斯,阿洛伊修斯 153
Lindstrom,Gary Edward 林斯特龙,加里·爱德华 538,539
Line printer 行式打印机 132—133
Lineal chart 直系图 295—296
Linear extensions 线性扩展,见 Topological sorting
Linear lists 线性表 222,227—292
Linear ordering 线性序 18,248,256
  embedding a partial ordering into 把偏序嵌入线性序,见 Topological sorting 248
  of binary trees 二叉树的线性序 316
  of trees 树的线性序 329
Linear probing 线性探测 423
Linear recurrence 线性递归(推) 80,85
Link 链接 223—226
  diagram 链接的图式 224
  field,purpose 链接字段的作用 407—408,434
  manipulation,avoiding errors in 在链接处理中避免出错 246—247
  null 空链接 224—225
Link variable 链接变量 224
Linked allocation of tables 表格的链接分配 224,240—242
  arrays 数组的链接分配 287—292
  contrasted to sequential 链接分配同顺序分配对照 240—242,283,407
  history 链接分配的历史 429—433
  linear lists 线性表的链接分配 224—226,249—251,255
  tree structures 树结构的链接分配 317,334—338
Linked memory philosophy 链接内存原理 241,409
Linking automaton 链接自动化 434—435
Linsky,Vladislav Sergeevich (Линский,Владиславсергеевич)林斯基,弗拉季斯拉夫·谢尔盖耶维奇 441
LISP LISP 语言 223,432—433
LISP 2 garbage collector LISP 2 废料收集程序 582
List 表:0 个或多个元素的有序序列
  circular 循环表 258—265,287,337,387,431
  doubly linked 双重链接表 265—266,273—277,283—284,339,387,414,416,424,431
  linear 线性表 221,227—292
  of available space 可用空间表,见 Available space list
List (capital-List) structures 列表结构 299—300,385—397,432—434
  copying 复制列表结构 397
  diagrams of 列表的图式 299—301,385—386
  distinguished from lists 同表的区别 222,387
  equivalence between 列表间的等价性 397
  notations for 列表的记号 299—301,385—386
  representation of 列表的表示 386—388,396
  sequential allocation 列表的顺序分配 396
List head,in circular lists 循环表的表头 260,287—288
  in Lists 列表的表头 386,390,392
  in threaded binary tree 在穿线二叉树中的表头 307,315
  in threaded trees 在穿线树中的表头 320
List processing systems 列表处理系统 222,388,432—434
Listing,Johann Benedict 利斯汀,约翰·本尼迪克 383
Literal constants in MIXAL MIXAL 中的文字常数 144,150
Literate Programming 文字化程序设计 190
Littlewood,John Edensor 利特尔伍德,约翰·艾登索尔 382
LLINK:Link to the left LLINK:对左边的链接 265—266,273—277
  in binary trees 二叉树中的 LLINK 302,305,310,316,431
  in Lists 在列表中的 LLINK 387
  in trees 在树中的 LLINK 321,331,346,358
LLINKT LTAG-LLINK 308
Lloyd,Stuart Phinney 劳埃德,斯图尔特·菲尼 175,176
Loading operators of MIX MIX 的装入操作符 125,135,148
Loading routine 装入程序 140,257—258
LOC LOC 语句 225—226
Local symbols of MIXAL MIXAL 中的局部符号 144—145,151
Locally defined function in tree 树中局部地定义的函数 333,342

Location 单元：一个计算机字或节点的内存地址，或内存单元本身

Location counter in MIXAL MIXAL 中的地址计数器 150

Location field of MIXAL line MIXAL 行的地址字段 140,147

Logan, Benjamin Franklin(= Tex), Jr. 小洛根，本杰明·富兰克林(= 特克斯) 71

Logarithms 对数 20—23
 binary 二进制对数 21,23
 common 常用对数 21,23
 natural 自然对数 21,23
 power series 幂级数的对数 87

Loop in a directed graph 有向图中的循环：从顶点到它自身的有向边 351

Loopstra, Bram Jan 卢珀斯特拉，布拉姆·简 220

Louise Wilhelmine Friederike Karoline Auguste Julia von Hesse-Kassel 路易斯·威廉明·弗里德里克·卡罗林·奥古斯特·朱莉娅·冯·赫西-卡塞尔 295,296

Lovász, László 洛瓦茨，拉茨洛 461

Lovelace, Augusta Ada Byron King, Countess of 洛夫莱斯，奥古斯塔·艾达·拜伦·金伯爵夫人 1

LTAG LTAG 字段 306,316,331—332,334,340,358

Lucas, Francois Édouard Anatole 卢卡斯，弗兰索伊斯·埃杜瓦德·阿纳托尔 66,77,78,83,258,464
 numbers L_n 卢卡斯数 465

Luhn, Hans Peter 卢恩，汉斯·彼得 429

Lukasiewicz, Jan 鲁卡西维兹，简 321

Lunch counter problem 便餐台问题 427

Luo, Jianjin 罗见今 383

Lynch, William Charles 林奇，威廉·查尔斯 555

Machine language 机器语言：当由一计算机线路解释时，支配此计算机动作的语言 Ⅷ—Ⅹ,120
 symbolic 符号机器语言，见 Assembly language 140

MacMahon, Percy Alexander 麦克马洪，珀西·亚历山大 460,559

Macro instruction 宏指令：在一个程序内可能经常被重复的一组指令或伪操作符的详细说明

Madnick, Stuart Elliot 马德尼克，斯图尔特·埃利奥特 433

Magic square 幻方，纵横图 156

Magnetic tape 磁带 131—132,435

Mailloux, Barry James 梅劳克斯，巴里·詹姆斯 433

malloc 见 Dynamic storage allocation

Mallows, Colin Lingwood 马洛斯，科林·林伍德 508

Margolin, Barry Herbert 马戈林，巴里·赫伯特 422

Mark I calculator (Harvard) 马克 I 计算器(哈佛) 219

Mark I computer (Ferranti) 马克 I 计算机(费兰蒂) 16

Mark bits 标记位 389—390

Marking algorithms 标记算法：标记由某些给定节点可访问的所有节点的算法 256—257,390—397

Markov, Andrei Andreevich (Марков, Андрей Андреевич), the elder 马尔可夫，安德烈·安德烈维奇(老的) 465
 chain 马尔可夫链
 process 马尔可夫过程 240,359—361

Markov, Andrei Andreevich (Марков, Андрей Андреевич) the younger 马尔可夫，安德烈·安德烈维奇(年轻的) 7

Markowitz, Harry Max 马柯维兹，哈里·马克斯 431

Markowsky, George 马柯维斯基，乔治 77,380

Martin, Alain Jean 马丁，阿兰·琼 574

Martin, Johannes Jakob 马丁，约翰尼斯·雅各布 583

Math. Comp.: *Mathematics of Computation* (1960—), a publication of the American Mathematical Society since 1965; founded by the National Research Council of the National Academy of Sciences under the original title *Mathematical Tables and Other Aids to Computation* (1943—1959) 《计算数学》杂志，美国数学协会出版

Math. Zeitschrift 《数学杂志》

Mathematical induction 数学归纳法 9—18,29,300,444
 generalized 广义数学归纳法 17

Matiyasevich, Yuri Vladimirovich (Матиясевич, Юрий Владимирович) 马蒂亚谢维奇，尤里·维拉季米诺维奇 82

Matrix 矩阵：二维数组 284—285,299
 Cauchy 柯西矩阵 34—35,444
 characteristic polynomial of 矩阵的特征多项式 469
 combinatorial 组合矩阵 33—34,559
 determinant of 矩阵的行列式 33—35,78,357—358,360

extendible 可扩充矩阵 292
Hilbert 希尔伯特矩阵 35
incidence 伴随矩阵 255
inverse of 矩阵的逆 33—34,70,292
multiplication 矩阵乘法 291
permanent of 矩阵的积和式 47
representation of 矩阵的表示 152—153,284—292
singular 奇异矩阵 292
sparse 稀疏矩阵 288—291
transpose of 矩阵的转置 174
triangular 三角矩阵 286,290
tridiagonal 对角矩阵 292
unimodular 单模矩阵 571
Vandermonde 范德蒙德矩阵 33—34,445
Matrix (Bush), Irving, Joshua 马特里克(布什),欧文·乔舒亚 31,32
Matrix tree theorem 矩阵树定理 357—358,556
Mauchly, John William 莫奇利,约翰·威廉 220
Maurolico, Francesco 莫罗利科,弗朗西斯科 15
Maximum, algorithm to find 求极大值的算法 92—97,141,178
Maximum norm 极大范数 102
McCall's 麦考尔的 VI
McCann, Anthony Paul 麦卡恩,安东尼·保罗 583
McCarthy, John 麦卡锡,约翰 431—432
McEliece, Robert James 麦克尔利斯,罗伯特·詹姆斯 446
McIlroy, Malcolm Douglas 麦基尔洛伊,马尔科姆·道格拉斯 546,552
McKeeman, William Marshall 麦基曼,威廉·马歇尔 220
Mealy, George 米利,乔治 433
Mean value 均值,见 Expected value
Meek, Hormer Vergil 米克,霍默·维吉尔 220
Meggitt, John Edward 梅吉特,约翰·爱德华 442
Melville, Robert Christian 梅尔维尔,罗伯特·克里斯琴 511
Memory 存储器,内存:一个计算机系统用来存储数据的部分 122
cell of 内存单元 123
hierarchy 内存分级 190,395,409,434
map 内存映像 409—410
types of 内存类型 227
Merner, Jack Newton Forsythe 默纳,杰克·牛顿·福赛思 216
Merrett, Timothy Howard 梅里特,蒂莫西·霍华德 532
Merrington, Maxine 梅林顿,马克辛 62
METAFONT METAFONT 软件 IV,XI,580,630
METAPOST METAPOST 软件 XI,630
Methodus Differentialis 《微分方法》
Military game 军棋 258,521
Miller, Kenneth William 米勒,肯尼思·威廉 119
Ming, An-T'u 明安图 383
Minimum path length 极小通路长度 376—382
Minimum spanning tree 极小生成树 351
Minimum wire length 极小导线长度 351
Minsky, Marvin Lee 明斯基,马文·李 398
Mirsky, Leon 米尔斯基,利昂 557
MIT Press 麻省理工学院出版社
Mitchell, William Charles 米切尔,威廉·查尔斯 497
MIX computer MIX 计算机 VIII—X,120—140
assembly language for MIX 计算机的汇编语言 140—151
extensions to MIX 计算机的扩充 139,218,238—239,427
instructions, summary MIX 计算机指令,小结 136—137
simulator of MIX 计算机的模拟程序 193—204
MIXAL: MIX Assembly Language MIX 的汇编语言 140—151,225—226
Mixed-radix number system 混合进制系统 286
Mixture of probability distributions 概率分布的混合 102
MMIX computer MMIX 计算机 120,179,206,308
Mock, Owen Russell 莫克,欧文·拉塞尔 221
mod 模 35—36
modulo 模 36
Mohammed, John Llewelyn 穆哈默德,约翰·卢埃林 499
Moivre, Abraham de 莫伊弗里,亚伯拉罕·德(棣莫弗) 71,80,83,102,174,508
Moments of probability distributions 概率分布的矩 101
Monitor routine 监督程序,见 Trace routine 203
Monte Carlo method 蒙特卡罗方法:以随机数进行实验 242,418—420
Moon, John Wesley 穆恩,约翰·韦斯利 383

Moore School of Electrical Engineering 穆尔电子工程学校 220
Mordell, Louis Joel 莫德尔,路易斯·乔尔 448
Morris, Francis Lockwood 莫里斯,弗朗西斯·洛克伍德 16,583
Morris, Joseph Martin 莫里斯,约瑟夫·马丁 538
Morrison, Emily Kramer 莫里森,埃米莉·克莱默 219
Morrison, Philip 莫里森,菲利普 219
Moschopoulos, Manuel 莫斯科普洛斯,曼纽尔 156
Moser, Leo 莫泽,利奥 62
Mother, in a tree structure 在树结构中的母亲 296
Motzkin, Theodor Samuel 莫特金,西奥多·塞缪尔 83,563
MOVE MOVE 指令 131,135,191,203
MOVE CORRESPONDING MOVE CORRESPONDING 语句 401,404—405,408
MUG: MIX User's Group, MIX 用户组 612
MUL (multiply) MUL 指令(乘法) 127—128,199,339,398—408,430
Multilinked structures 多重链接结构 222,274—275
Multilist representation 多重表的表示 287
Multinomial coefficients 多项式系数 61,372
Multinomial theorem 多项式定理 61
Multipass algorithm 多遍扫描算法 189—190,191—192
Multiple 倍数:如果 y 是 x 的因子,即对于某个整数 k,有 $x = ky$,则 x 是 y 的倍数
Multiple entrances to subroutines 对子程序的多个入口 180
Multiple exits from subroutines 从子程序的多个出口 181,255
Multiple precision arithmetic 多精度算术 192
Multiple precision constants 多精度常数 588—590
Multiple summation 多重和 29—32
Multiplication of permutations 排列的乘法 158—161,165—166,359
Multiplication of polynomials 多项式的乘法 262,265,342
Multiplication of sparse matrices 稀疏矩阵的乘法 291
Multiplicative function 乘性函数 38—39
Multiset 多重集:和集合类似,但其元素可以出现一次以上
Multiway decisions 多路判断 153
Munro, James Ian 芒罗,詹姆斯·伊恩 495
Nagorny, Nikolai Makarovich (Нагорный, Николай Макарович) 纳戈尔内,尼古拉·马卡罗维奇 8
Nahapetian, Armen 内和彼蒂安,阿尔门 549
Napier, John, Laird of Merchiston 内皮尔,约翰,麦琪斯顿的地主 20
Nash, Paul 纳什,保罗 528
National Science Foundation 国家科学基金会[美] X
Natural correspondence between binary trees and forests 二叉树和森林之间的自然对应 317—318,329
Natural logarithms 自然对数 20,24
Natural Physics Laboratory [伦敦]自然物理实验室
Naur, Peter 瑙尔,彼得 15
Needham, Joseph 尼达姆,约瑟夫 56
Neely, Michael 尼利,迈克尔 424
Negative 负:小于0(非0)
Negative binomial distribution 负二项分布 102
Nested parenthesis 嵌套的括弧 297—298,331,567
Nested sets 嵌套的集合 298,301
Nesting store 嵌套存储 229
Network 网络:含附加数据,比如边或顶点上的权的图
Neumann, John von (= Margittai Neumann János) 诺伊曼,约翰·冯(=马吉泰·纽曼·雅诺什),一般译作冯诺伊曼 15,219,221,429
Neville, Eric Harold 内维尔,埃里克·哈罗德 561
Newell, Allen 纽厄尔,艾伦 220,429—430,432
Newton, Isaac 牛顿,艾萨克 20,54,466
  identities 牛顿恒等式 466
Next-fit method of storage allocation 存储分配的第二适合方法 421,425,587
Nicolau Alexandru 尼古劳,亚历山德鲁 583
Nicomachus of Gerasa 格拉萨的尼科梅彻斯 17
Nielsen, Norman Russell 尼尔森,诺曼·拉塞尔 422
Nieuw Archief voor Wiskunde 《科学新档案》
Nil link 空链接,见 Null link
Niven, Ivan Morton 尼文,伊凡·莫顿 74
Noah, son of Lamech 诺亚,拉默之子 295
Node 节点:数据结构的基本组成部分 223,434—435
  address of 节点的地址 223
  diagram of 节点的图式 224

link to 对节点的链接 223
notations for field 字段的节点记号 225—227,430
size of 节点的大小 244,285,409,425
NODE NODE字段 226
Node variable 节点变量 226
Noncrossing partitions of a polygon 多边形的不相交划分(分划) 384
Nonnegative 非负:零或正
NOP (no operation) NOP指令(空操作) 131
Normal distribution 正态分布 100,118
approximately 近似正态分布 101—102
Notations, index to 符号索引 592—595
Novi *Comment. Acad. Sci. Pet.* 《彼得堡科学院最新公报》
Null link(Λ) 空链接 224—225
in binary trees 二叉树中的空链接 305,314
in diagrams 图表中的空链接 224
in trees 树中的空链接 302
NUM (convert to numeric) NUM指令(转换成数值) 134
Number definitions 数的定义 19
Number system 数系:表示数的一种语言
binary 二进数系 22—24
combinatorial 组合数系 70,531
decimal 十进数系 19,70
Fibonacci 斐波那契数系 83,464
mixed-radix 混合进制数系 286
octal 八进数系 589
phi φ数系 84
Number theory, elementary 初等数论 37—41
Nygaard, Kristen 尼加德,柯里斯顿 219,433

O-notation (大)*O*记号(符号) 103—106,114
O'Beirne, Thomas Hay 奥贝恩,托马斯·海 489
Octal values of constants 常数的八进制值 589
Odlyzko, Andrew Michael 奥德利兹柯,安德鲁·迈克尔 117,536
Oettinger, Anthony Gervin 奥廷格,安东尼·杰文 432
Office of Naval Research 海军研究所[美] x,220
Okada, Satio 冈田幸雄,后称冈田幸千生 553
Oldenburg, Henry 奥尔登伯格,亨利 54
Oldham, Jeffery David 奥德海姆,杰弗里·戴维 xi
Omphaloskepsis 意守丹田 205
One address computer 一地址计算机 123,332
One-way equalities 单向相等性 104
One-way linkage 单向链接,见 Circular linkage, Straight linkage
Onodera, Rikio 小野寺力夫 552
Open subroutine 开子程序,见 Macro instruction 219
Operation code field, of MIX instruction MIX指令的操作码字段 123
of MIXAL line MIXAL语句的操作码字段 140,150
Optimal search procedure 最优检索(查找)过程 379
Order of succession to the throne 御座之沿袭次序 318
Ordered trees 有序树,见 trees 293—294,353
enumeration of 有序树的枚举 366—367,375,383
Ordering 次序:一个集合的对象之间的传递关系
lexicographic 字典序 18,285—286,291
linear 线性序 18,247,256
linear, of tree structures 树结构的线性序 315,329
partial 偏序 247—248,255—256,329,534,546
well 良序 18,316
Oresme, Nicole 奥雷斯姆,尼科尔 20
Oriented binary trees 有向二叉树 373—374
Oriented cycle in a directed graph 有向图的有向循环 351
Oriented forests 有向森林 335—337
Oriented path in a directed graph 有向图的有向通路 351
Oriented subtrees, enumerated 枚举的有向子树 357
Oriented trees 有向树 293—294,297,351—360
canonical representation 有向树的典型表示 367—371,374—375,560—561
converted to ordered trees 有向树转变成有序树 330
defined 确定的有向树 352
enumerated 枚举的有向树 364—365,366—372,382
represented in computer 在计算机中表示的有向树 330,335,356
with root changed 改变根的有向树 356
ORIG (origin) ORIG伪指令 141,146,150
Orlin, James Berger 奥林,詹姆斯·伯格 554
Orthogonal lists 正交表 284—292
Orthogonal vectors of permutations 排列的正交向量 177

Osaka 大阪[日]
Otoo, Ekow Joseph 奥图,埃考·约瑟夫 532
Otter, Richard Robert 奥特,理查德·罗伯特 372,559
OUT (output) OUT 指令(输出) 132,216
Out-degree of a vertex 向量的出度 351
Outout 输出 4,206—218
  buffering 输出缓冲 142,206—218,221
  operators of MIX MIX 的输出操作符 131—133
Output-restricted deque 输出受限双端队列 221—231,255,259
Overflow Overflow 指令(溢出) 233—238,243—245,254—255
Overflow toggle of MIX MIX 的溢出开关 122,130,137,199,205,218

Packed data 打包数据:已经被压缩到一个小空间中的数据,例如,把两个或多个数据元素置入内存同一字中 124,152
Paging 分页 424
Pallo, Jean Marcel 保罗,琼·马塞尔 548
Palm tree 棕榈树 551
Paper type 纸型 132—133,219,221
Parallelism 并行性 282
Parameters of subroutines 子程序的参数 179,181,219
Parent, in a tree structure 树结构中的双亲(父亲) 296,301,317—318
  in a threaded tree 在穿线树中的双亲(父亲) 537
Parent links 父链接 329,335—337,340—341,353,356,402—407
Parentheses 括号(弧) 297—298,331,567
Parker, Douglass Stott, Jr. 小帕克,道格拉斯·斯托特 566
Parmelee, Richard Paine 帕米利,理查德·佩因 422
Partial field designations in MIX MIX 中的部分字段的指定 122—124,135,139,197
Partial fractions 部分分式 59—60,70,79
Partial ordering 偏序 247—248,255—256,329,534,545
Partitions of a set 一个集合的分划 71,451
Partitions of an integer 一个整数的分划 10,31,90
  generating function 整数分划的生成函数 84,91
Pascal, Blaise 帕斯卡,布莱斯 15,49
  triangle 帕斯卡尔三角,见 Binomial coefficients 49—50,66,68,70,81,469
Pass, in a program 一个程序中的趟(遍) 189—190
Patashnik, Qren 帕塔什尼克,奥伦 9
Path, in a graph or directed graph 在一个图或有向图中的通路 343
  oriented 有向通路 351
  random 随机通路 358—359
  simple 简单通路 343,349,351,356
Path compression 通路压缩 546
Path length of a tree structure 一个树结构的通路长度 376—382
  average 平均通路长度 381
Patienee (solitaire) 独玩游戏(单人游戏) 356
Patt, Yale Nance 帕特,耶尔·南斯 479
Pawlak, Zdzislaw 波莱克,泽吉斯洛 432
PDP-4 computer PDP-4 计算机 120
Peck, John Edward L. 佩克,约翰·爱德华·L 433
Pedigree 家系 295—296
Peirce, Charles Santiago Sanders 皮尔斯,查尔斯·圣地亚哥·桑德斯 563
Penrose, Roger 彭罗斯,罗杰 557
Peripheral 外围设备,一个计算机系统的输入输出设备 132
Perlis, Alan Jay 珀利斯,艾伦·杰伊 305,431—432
Permanent of a square matrix 正方矩阵的积和式 47
Permutations 排列 41—42,47,94—95,158—178,230—231
  in place 排列的位置 8,159,177—178,495
  inverse of 排列的逆 102,168—171,175
  multiplication of 排列的乘法 159—160,165—166,350
  notations for 排列记号 158
  orthogonal vectors of 排列的正交向量 177
Persian mathematics 波斯数学 2
PERT network PERT 网络 247—248
Petkovšek, Marko 彼特科夫塞克,马可 61
Petolino Joseph Anthony, Jr. 小皮托利诺,约瑟夫·安东尼 487
Pfaff, Johann Friedrich 帕夫,约翰·弗里德里克 455
Pflug, Georg Christian 弗莱格,乔治·克里斯琴 418
Phi(ϕ), 见 Golden ratio 77
  number system ϕ数系 83
Phidias, son of Charmides 菲迪亚斯,查米迪斯之子 77

Philco S2000 computer 菲尔科 S2000 计算机 120
Phyllotaxis 叶序 77
Pi(π) 圆周率 19,588—589
Wallis's product for π的沃利斯积 48,112
Pingala, Ācārya 平加拉,阿卡里亚 49
Pile 堆 229
Pilot ACE computer 向导 ACE 计算机 220
Pipe 管道 189
Pipe line 管道 500
Pisano, Leonardo 皮萨诺,伦纳德 76—77,80
Pivot step 主元步 281—290,292
PL/I PL/I 语言 407—408
PL/MIX MIX 计算机的 PL/I 语言 150
Plane tree 平面树,见 Ordered tree 293
Plex 丛 430
Poblete Olivares, Patricio Vicente 波伯利特·奥利瓦里斯,帕特里西欧·维森特 495
Poincaré, Jules Henri 波因克尔,朱利斯·亨利 460
Pointer 指针,见 Link
Pointer machines 指针机器 344
Poirot, Hercule 波伊罗特,赫尔克里 XVII
Poisson, Siméon Denis, distribution 泊松,西米恩·丹尼斯分布 102,496
tail of 泊松分布的尾部 472
Polish notation 波兰记号,见 Prefix notation、Postfix notation
Polonsky, Ivan Paul 波伦斯基,伊凡·保罗 433
Pólya, György(= George) 波利亚,乔治 15,90,375,376,381,382,466
Polynomials 多项式 51,54,60,64,65,67,104
addition of 多项式加法 260—265,338—340
Bernoulli 伯努利多项式 40,109—110,473
differences of 多项式差 60
multiplication of 多项式乘法 262,265,341
representation of 多项式表示 260—261,265,337—338
Pool of available nodes 可用节点池,见 Available space list 244
Pooled buffers 联营缓冲 214,216
Pop up a stack 弹出栈,出栈:删去栈顶元素 229,230,232—233,235,241,246,255,258—259,264,430
Positive 正的:大于0的(非0)
Postfix notation 后缀记号 320,334,563
Posting a new item 插入一个新项,见 Insertion
Postorder for a binary tree 二叉树的后根序 302,304,313—315,329
Postorder for a tree 树的后根序 318—321,329,330,432
Postorder with degrees, representation of trees 树表示,带度数的后根序 333,341—342
PostScript 一种字体 192
Poupard, Yves 波帕德,维斯 568
Power of a number 数的幂 20
evaluation 幂的计算 479
Power series 幂级数:形如$\sum_{k\geq 0}a_k z^k$的和,见 Generating function
convergence of 幂级数的收敛性 84,374
manipulation of 幂级数的处理 114
Pratt, Vaughan Ronald 普拉特,沃恩·罗纳德 41,510,562
Prefix notation 前缀记号 320
Preorder for a binary tree 二叉树的先根序 303,305,312—314,329
Preorder for a tree 树的先根序 318—320,329,331,430
Preorder sequential representation of trees 树的先根序的顺序表示 332,343
with degrees 带度数的先根序 340,430
Prepostorder 先后根序 538
Prim, Robert Clay 普赖姆,罗伯特·克莱 351
Prime numbers 素数 16,37,41,43—44,47,66—67,82—83
algorithm to compute 计算素数的算法 142—144
factorization into 分解成素数因子 38
Princeton University Press 普林斯顿大学出版社
Prinz, Dietrich Günter 普林兹,迪特里希·冈特 220
Priority queue 优先队列 409,527,560
Probability distribution 概率分布:支配一个随机变量的值的概率的描述 94—103
average ("expected") value of 概率分布的均值(期望值) 94—99
variance of 概率分布的方差 94—99
Probability generating funtion 概率生成函数 99
Procedure 过程,见 Subroutine
Procedure for reading this set of books 阅读本套书的流程 XII—XIII,8
Prodinger, Helmut 普罗丁格,赫尔穆特 476

Profile of a program 程序的侧面：程序的每个指令执行的次数 141,164,205,283,500
Program 程序：一个计算方法在某个精确的形式化语言下的表示 5
Programming language 程序设计语言：用于写程序的精确的形式化语言
Programs, hints for construction of 对构造程序的提示 183—184,282
Progression, arithmetic, sum of 算术级数的和 9,28—29,52
Progression, geometric, sun of 几何级数的和 28,86
Proof of algorithms 算法的证明 4—5,11—15,304,342,397,409
Proof of termination 终止证明 14—15,16—18,364
Proper divisor 真因子，见 Divisor
Property 𝒜 𝒜特性 556
Propositional calculus 命题演算
Prosody 韵律学 49,77
Prüfer, Ernst Paul Heinz 普鲁弗，厄恩斯特·保罗·海因茨 384
Pseudo-operator 伪操作符：程序设计语言中用来控制把该语言翻译成机器语言的一种构造 141
Psi function $\psi(z)$ ψ函数 40,72,462
Purdom, Paul Walton, Jr. 小珀多姆，保罗·沃尔顿 420,422
Push down list 下推表，见 Stack 228
Push down a stack 下推一个栈：插入一个新的栈顶元素 229,230,232—233,235,242,244,255,258—259,263,430

q-nomial coefficients q项式系数 62,70,453,460
q-nomial theorem q项式定理 70,464
Quadratic Euclidean domains 二次欧几里得整环 438
Quadratic reciprocity law 二次倒数律 41
Quadruply linked binary tree 二次链接二叉树 316
Quadruply linked trees 二次链接树 339
Quadtrees 二次树 534
Qualification of names 名称的性质 398—409
Quasi-parallel processing 拟并行处理 283
Queue 队列 229—231,249—251,431,547,576
  deletion from the front 从队列的前端删除 230,232—233,242,247,250,258—259
  insertion at the rear 在队列的后端插入 230,232—233,240,246,250,258—259
  linked allocation 队列的链接分配 245—247,255,258—259,273
  sequential allocation 队列的顺序分配 232—233,239,240,242
Quick, Jonathan Horatio 奎克，乔纳森·霍雷肖 472
Quotient 商 36

Rahman, Mizanur 拉曼，米泽纽尔 460
Railway network 铁路网 228
Ramanan, Prakash Viriyur 拉曼南，普拉卡什·弗里耶 509
Ramanujan Iyengar, Srinivasa 拉曼纽恩·伊恩加尔，斯里尼瓦萨 10,117,118
Ramshaw, Lyle Harold 拉姆肖，莱尔·哈罗德 456
Ramus, Christian 拉马斯，克里斯琴 68
Randell, Brian 兰德尔，布赖恩 192,422
Random path 随机通路 359—360
Raney, George Neal 拉尼，乔治·尼尔 369,371,563
Raphael, Bertram 拉菲尔，伯特龙 431
Rational number 有理数 18,23,155
Rax, Yoav 拉兹，约弗 506
RCA 601 RCA 601 计算机 120
Reactive process 往复过程 4
Read, Ronald Cedric 里德，罗纳德·塞德里克 536
Reading 读：进行输入 206
Real number 实数 18
Real part of a complex number 复数的实部 18
Real-time garbage collection 实时废料收集 397
Reallocation of memory 内存再分配，也见 Compacting memory 234—238,424,429
Rear of queue 队列尾部 229
Recently used bit 最近使用过的位 424
Recipe 食谱，菜谱 5
Reciprocity formulas 互反公式，倒数公式 40—41
Recomp II computer 雷科姆普 II 计算机 120
Record 记录：连续数据的集合，也见 Node 223
Records, blocking of 记录的分块 210,216
Rectangular arrays 矩形数组 284—292
Recurrence relation 递归关系：借助于前边的元素定义一个序列的每个元素的一种规则 84—86
Recursion induction 递归归纳法 304,536,540
Recursive definition 递归定义 293,297,302,303,317,329,339
Recursive List 递归列表 300

Recursive use of subroutines 子程序的递归使用 182
Redheffer, Raymond Moos 雷德赫弗,雷蒙德·穆斯 461
Reeves, Colin Morrison 里弗斯,柯林·莫里森 418
Reference 访问,引用,见 Link 223
Reference, counters 访问计数器 389—390,396,433
Reflection principle 反射原理 508,563,568
Reflective laws 反射率 51,458
Reflexive relation 反身关系 247,334
Registers 寄存器:一台计算机的内部线路部分,要处理的数据放在其中。保持于一机器中的大多数可访问数据出现于它的寄存器中
of MIX MIX 的寄存器 121
saving and restoring contents of 保存和恢复寄存器的内容 179,189,204,218
Regular directed graph 正则有向图 358
Reingold, Edward Martin 雷恩戈德,爱德华·马丁 21,489
Relation 关系:对某些集合的元素(通常是对有序的对偶)成立的一个性质;例如,"<"是对于整数的有序对(x,y)定义的一个关系,且性质 $x<y$ 成立当且仅当 x 小于 y
antisymmetric 反对称关系 247
asymmetric 非对称关系 247
equivalence 等价关系 335—337,456
irreflexive 非反身关系 247
reflexive 反身关系,见 reflexive relation 247,335
symmetric 对称关系 335
transitive 传递关系,见 Ordering 104,247,335
Relative error 相对误差 112
Relatively prime integers 互素整数 36
Releasing a buffer 释放缓冲区 209—213,216—217
Remainder 余数 36,154
Remove from a structure 从结构中撤消,见 Deletion
Rényi, Alfréd 伦尼,艾尔弗雷德 562
Replacement operation(←) 替代运算 3
Replicative function 重叠函数 39—40
Representation (inside a computer) (在一计算机内的)表示
methods for choosing 选择表示的方法 227—228,399—407
of algebraic formulas 代数公式的表示 319,431
of binary trees 二叉树的表示 302,305,310,315—316
of deques 双端队列的表示 238,266,283
of forests 森林的表示 316
of Lists 列表的表示 386—389,395
of oriented trees 有向树的表示 330,335,356
of polynomials 多项式的表示 260—261,266,337—338
of queues 队列的表示 232—233,238—241,246—247,255,258—259,273
of stacks 栈的表示 232—241,245,255—256,258—259,315,392
of trees 树的表示 330—338,340—342,431—432
Reprogramming 重新编写程序 193
Reservation of free storage 自由存储的保留 243—245,276,410—412,416,424—427
Reversal of a string 串的反转 177
Reversing a list 反转表 255,265
Reversion storage 反转存储 229
Ribenboim, Paulo 里本博伊姆,保罗 436
Rice, Stephan Oswald 赖斯,斯蒂芬·奥斯瓦德 536
Richmond, Lawrence Bruce 里奇蒙德,劳伦斯·布鲁斯 536
Riemann, George Friedrich Bernhard, zeta function $\zeta(s)$ 黎曼,乔治·弗里德里克·伯恩哈德 $\zeta(s)$ 函数 73
Right subtree of a binary tree 二叉树中的右子树 297,302
Right-threaded binary trees 右穿线二叉树 310,315—316
Right-threaded trees 右穿线树 320,359
Right-to-left maximum or minimum 自右自至的极大值或极小值 94—98,100—102,172
Ring structure 环形结构 337—338
Riordan, John 赖尔登,约翰 374,463,565
RISC: Reduced Instructon Set Computer 精减指令集计算机 120
Rising factorial powers 升阶幂级数 46,68,593
Ritchie, Dennis MacAlistair 里奇,丹尼斯·麦卡利斯特 433
RLINK RLINK 字段:对右边的链接 265—266,273—276
in binary trees 二叉树中的 RLINK 302,305,310,315,431
in Lists 列表中的 RLINK 386,387
in trees 树中的 RLINK 321,330—335,340,359

RLINKT RLINKT字段 308
Robinson, Raphael Mitchel 鲁宾逊, 拉菲尔·米切尔 557
Robson, John Michael 罗伯逊, 约翰·迈克尔 421, 424, 427, 538, 573, 586
Rodrigues, Banjamin Olinde 罗德里格斯, 班杰明·奥林德 383
Roes, Piet Bernard Marie 罗斯, 皮特·伯纳德·玛丽 91
Rogers, Leonard James 罗杰斯, 伦纳德·詹姆斯 459
Rokicki, Tomas Gerhard 罗基基, 托马斯·格哈特 193
Roll 卷形物 229
Root of a directed graph 有向图的根 351
Root of a number 数的根 20, 23
Root of a tree 树的根 107, 293, 394, 301
  change of 树的根的变动 356
Rooting a free tree 求自由树的根 352
Roots of unity 单位根 86
Rosenberg, Arnold Leonard 罗森伯格, 阿诺德·伦纳德 531
Rosenstiehl, Pierre 罗森斯蒂尔, 皮埃尔 231
Ross, Douglas Taylor 罗斯, 道格拉斯·泰勒 422, 430, 433
Rotating memory devices 翻转存储设备 218, 429
Rothe, Heinrich August 罗瑟, 海因里希·奥古斯特 59, 455
Rounding 舍入 38, 80, 154, 176
Rousseau, Cecil Clyde 鲁索, 塞西尔·克莱德 477
Roving pointer 游动指针 576
Row major order 行优先次序 153, 175, 285
RTAG RTAG字段 305, 315—316, 320, 331—333, 340, 359
Running time 运行时间, 见 Execution time
Russell, David Lewis 拉塞尔, 戴维·刘易斯 582
Russell, Lawford John 拉塞尔, 劳福德·约翰 192

Saddle point 鞍点 153
Salton, Gerard Anton 萨尔顿, 杰勒德·安东 333, 431
Sammet, Jean Elaine 萨米特, 琼·伊莱恩 443, 545
Satterthwaite, Edwin Hallowell, Jr. 小萨特思韦特, 埃德温·霍洛韦尔 220
Saving and restoring registers 保存和存储寄存器 179, 189, 194, 218
Schöffer, Alejandro Alberto 谢弗, 亚历山德罗·艾伯托 485
Schatzoff, Martin 沙佐夫, 马丁 422
Scherk, Heinrich Ferdinand 谢尔克, 海因里希·费迪南德 459
Schlatter, Charles Fordemwalt 施拉特, 查尔斯·福登沃尔特 431
Schlatter, William Joseph 施拉特, 威廉·约瑟夫 431
Schlesvig-Holstein-Sønderborg-Glücksborg, Christian von 施莱斯维格-霍尔斯坦-桑德伯格-格卢克斯伯格, 克里斯琴·冯, 见 Christian IX
Scholten, Carel Steven 斯科尔顿, 卡莱尔·史蒂文 221, 574
Schoor, Amir 斯库尔, 艾米尔 531
Schorr, Herbert 肖尔, 赫伯特 393, 397
Schreiber, Peter 施赖伯, 彼得 78
Schreier, Otto 施利耶尔, 奥托 363
Schröder, Friedrich Wilhelm Karl Ernst 施罗德, 弗里德里克·威廉·卡尔·厄恩斯特 562
Schwartz, Eugene Sidney 施瓦茨, 尤金·西德尼 479
Schwartz, Karl Hermann Amandus, inequality 施瓦茨, 卡尔·赫尔曼·阿曼达斯不等式 32
Schwenk, Allen John 施文克, 艾伦·约翰 465
Schweppe, Earl Justin 施韦普, 厄尔·贾斯廷 431
SCOPE link SCOPE链接 332, 342, 409
Scroll 卷形物 228
Segner, Johann Andreas von 西格纳, 约翰·安德烈·冯 384, 507
Seki, Takakazu 关孝和 108, 110
Self-modifying code 自修改代码 179, 191
Selfridge, John Lewis 塞尔弗里齐, 约翰·刘易斯 75
Semaphore 信号灯 221
Semi-invariants of a probability distribution 一个概率分布的半不变量 99—102
Sentinel 标志: 放置在一个表格中的特殊值, 例如, 用于标志此表格的边界, 以便伴随的程序易于认识 208—209, 261, 538
Sequential (consecutive) allocation of tables 表格的顺序(连续的)分配 232
  arrays 数组的顺序分配 132—133, 285—287, 290—292
  contrasted to linked 顺序分配同链接分配对照 241—243, 282, 401
  history 顺序分配的历史 429

linear lists 线性表的顺序分配 232—241,250—252,308
tree structures 树结构的顺序分配 330—334,340—343,408
Series,infinite 无穷级数:一个无穷的求和
Series-parallel networks 平行级数网络 559
Sets,partition of 集合的分划 72,451
Sha,Jichang 沙基昌 518
Shakespeare(= Shakspere) William 莎士比亚·威廉 212,436
Shams Baragh,Armin 沙姆斯·巴拉,阿明 498
Shaw,John Clifford 肖,约翰·克利福德 220,429—430
Shelf 架子 229
Shephard,Geoffrey Colin 谢泼德,杰弗里·科林 363,557
Shepp,Lawrence Alan 谢普,劳伦斯·艾伦 175,176
Shift operators of MIX MIX的移位操作符 131,203
Shor,Peter Williston 肖尔,彼得·威尔斯顿 485
Shore,John Edward 肖尔,约翰·爱德华 418,422
Shylock 谢洛克 436
Sibling,in a tree structure 树结构中的兄弟 296
Sibling link 兄弟链接,也见 RLINK in trees 316—318,401—407
SICOMP: SIAM Journal on Computing, pubished by the Society for Industrial and Applied Mathematics since 1972 《SIAM计算杂志》:[美]工业与应用数学学会出版的杂志
Sideways addition 横向加法,横加 127,449
Sign function (sign x) 符号函数 444
Silver,Roland Lazarus 西尔弗,罗兰德·拉扎勒斯 499
Similar binary trees 相似的二叉树 310—312
Similar forests 相似的森林 328
Simon,Herbert Alaxander 西蒙,赫伯特·亚历山大 220,429—430
Simonovits,Miklós 西蒙诺维茨,米克洛 475
Simple oriented path 简单有向通路 351,355
Simple path 简单通路 343,349
SIMSCRIPT language SIMSCRIPT语言 431
SIMULA I language SIMULA I语言 219
Simulated time 模拟时间 267,274
Simulation 模拟:某个系统的模仿 418
continuous 连续模拟 266,285
discrete 离散模拟 193,266—285
of one computer on another 在一台上计算机上模拟另一台计算机 8,192—193
of one computer on itself 在一台计算机上模拟它本身 204—206
Singh,Parmanand 辛格,帕曼南德 77
Singleton cycle of a permutation 排列的单个循环 157,164,172—173
Singular matrix 奇异矩阵 292
Singularity of a function 函数的奇异性 373
Sister,in a tree structure 树结构中的姐妹 296
SLA (shift left rA) SLA指令(左移 rA) 131,502
SLAX (shift left rAX) SLAX (左移 rAX) 131,502
SLC (shift left rAX circularly) SLC 指令(循环左移 rAX) 131,502
SLIP SLIP程序设计语言 432—433
Sloane,Neil James Alexander 斯隆,内尔·詹姆斯·亚历山大 564
Smallest-in,first-out 最小的进,先出 527
SNOBOL SNOBOL语言 433
SODA: Proceedings of the ACM - SIAM Symposia on Discrete Algorithms, inaugurated in 1990 《ACM-SIAM离散算法会议论文集》,1990年创办
Software 软件:扩充计算机硬件能力的通用程序
Solitaire (patience) 单人(独玩)游戏 356—357
Son,in a tree structure 在一个树结构中的儿子 296
Soria,Michèle 索里亚,米歇尔 471
Spanning subtrees 生成子树 345—350
minimum cost 最小代价生成子树 351
Sparse array trick 稀疏数组技巧 292
Sparse matries 稀疏矩阵 288—291
Sparse-update memory 较少更新存储 285
Speedcoding 快速编码 220
Spieß,Jürgen 斯皮伯,朱尔金 88
Spine of a binary tree 二叉树的中心 540
SRA (shift right rA) SRA 指令(右移 rA) 131,502
SRAX (shift right rAX) SRAX 指令(右移 rAX) 131,502
SRC (shift right rAX circularly) SRC 指令(循环右移 rAX) 131,502
ST1(store rI1) ST1 指令(存储 rI1) 126,201
STA (store rA) STA 指令(存储 rA) 126,201
Stack 栈 228—231,303—304,306,308—309,333,342,390—391,396,403—404,430—431

deletion("popping") 栈的删除("弹出") 230, 231, 232—233, 235, 241, 246, 255, 258—259, 263, 430
insertion("pushing") 栈的插入("下推") 230, 231, 232—233, 235, 241, 245, 255, 258—259, 430
linked allocation 栈的链接分配 245, 255, 256, 258—259, 315, 392
pointer to 栈的指针 232, 245
sequential allocation 栈的顺序分配 232—241, 308
Stack permutations 栈排列 231, 314
Standard deviation of probability distribution 概率分布的标准差：方差的平方根，预料一个随机变量对它的均值可以有多大偏离的标志 94
Stanford University 斯坦福大学 Ⅱ, Ⅹ, 282, 525
Stanley, Richard Peter 斯坦利，理查德·彼得 383, 563, 568
Staudt, Karl Georg Christian von 斯托特，卡尔·乔治·克里斯琴·冯 382
Steady state 稳定状态 359—360
Stearns, Richard Edwin 斯特恩斯，理查德·埃德温 436
Steele, Guy Lewis, Jr. 小斯蒂尔，盖伊·刘易斯 574
Steffens, Elisabeth Francisca Maria 斯蒂芬斯，伊丽莎白·弗朗西斯卡·玛丽亚 574
Steffensen, Johan Frederik 斯蒂芬森，约翰·弗雷德里克 473
Stevenson, Francis Robert 史蒂文森，弗朗西斯·罗伯特 549
Stickelberger, Ludwig 斯蒂克尔贝格，路德维格 47
Stigler, Stephen Mack 施蒂格勒，斯蒂芬·麦克 421, 422
Stirling, James 斯特林，詹姆斯 43—45, 63, 66, 71, 84, 110, 173
approximation 斯特林近似公式 47, 110—111
Stirling numbers 斯特林数 62—65, 68—72, 75, 94—95, 476, 554
asymptotic behavior 斯特林数的渐近特性 62
combinatorial interpretations 斯特林数的组合解释 62, 72, 172
duality law 斯特林数的对偶律 64
generating functions 斯特林数的生成函数 87
modulo *p* 模 *p* 斯特林数 461
table 斯特林数表 62
STJ(store rJ) STJ 指令(存储 rJ) 126, 179, 200
STOC: Proceedings of the ACM Symposia on Theory of Computing, inaugurated in 1969 《ACM 计算理论会议论文集》，1969 年创办
Stolarsky, Kenneth Barry 斯托拉斯基，肯尼思·巴里 464
Storage allocation 存储分配：选择用于存储数据的存储单元，见 Available space list, Dynamic storage allocation, Linked allocation, Sequential allocation
Storage mapping function 存储映像函数：给定数组节点的下标，其值为此数组节点的位置的函数 285—287, 290—292
Store 存储："memory"的英国用法
Storing operators of MIX MIX 的存储操作符 126, 200
Straight linkage 直接链接 241, 245—246, 387
String 串：0 个或多个符号的有限序列，见 Linear lists 7, 84, 176, 259, 464
binary 二进串 568—569
concatenation 串的连接 259
manipulation 串的处理 433, 434
Strong, Hovey Raymond, Jr. 小斯特朗，霍维·雷蒙德 531
Strongly connected directed graph 强连通有向图 351, 356
Structure, how to represent 如何表示结构 227, 399—407, 434
Struik, Dirk Jan 斯特罗伊克，德克·简 53, 450, 466
Stuart, Alan 斯图尔特，艾伦 95
STX (store rX) STX 指令(存储 rX) 126, 200
STZ (store zero) STZ 指令(存储零) 126, 200
SUB (subtract) SUB 指令(减法) 127—128, 199
Subadditive law 次加律 585
Subi, Carlos Samuel 萨比，卡罗斯·塞缪尔 499
Subroutine 子程序 152, 178—188, 192, 197—198, 202, 255, 264—265, 275—276, 323
allocation of 子程序的分配 257—258
history 子程序的历史 219—220
linkage of 子程序链接 178, 219
Subscript 下标，见 Index 2—3
Substitution operation(←) 替代运算 2
Subtrees 子树 293
average size of 子树的平均大小 381
free, enumeration of 自由子树的枚举 357—358
Summation 求和 25—35
by part 分部求和 40, 73—75

Euler's formula 欧拉求和公式 107—112,116,119
interchange of order 交换求和次序 27,29,31,39
multiple 多重求和 29—32
of arithmetic progression 算术级数求和 9,27—28,52
of binomial coefficients 二项式系数求和 52—72,73—75,83,92
of geometric progression 几何级数求和 27,84
of powers 幂的和 111
related to integration 求和同积分的关系 107—112
Sun SPAR Cstation Sun SPARC 工作站 630
Supremum 最小上界 33
Suri, Subhash 苏里,萨布哈什 485
Sutherland, Ivan Edward 萨瑟兰,伊凡·爱德华 430
Swainson, William 斯温森,威廉 313
Swapping buffers 转储缓冲 142,153,208—209,216
Swift, Charles James 斯威夫特,查尔斯·詹姆斯 221
Swift, Jonathan 斯威夫特,乔纳森 596
Switching table 开关表 152,196—197,200,201,502
Sylvester, James Joseph 西尔威斯特,詹姆斯·约瑟夫 383,443,553,556
Symbol manipulation 符号处理,用于数据处理的一般术语,通常可应用于诸如串或代数公式的非数值处理
Symbol table algorithms 符号表算法 168,400
Symbolic machine language 符号机器语言,见 Assembly language
Symmetric functions 对称函数 88—90,466
elementary 初等对称函数 34,90,466
Symmetric order for a binary tree 二叉树的对称序 303—307,313—315,329
Symmetric relation 对称关系 335
Synchronous discrete simulation 同步离散模拟 266,284
System 系统:相互连接或彼此相互作用的对象或方法的集合
System/360 computers 系统/360 计算机 120,180,220
Szekeres, George 泽克勒斯,乔治 565
Szpilrajn, Edward 斯皮尔拉因,爱德华 254

t-ary trees t 叉树 316,381
enumeration of t 叉树的枚举 374,563
sequential allocation t 叉树的顺序分配 378
Table-driven program 表格驱动程序,见 Interpreter,Switching table
Tables, arrangement of, inside a computer 表格,一台计算机内的表格安排,见 Representation
Tables of numerical quantities 数值量表 50,62,589—590
Tag field in tree node 树节点中的标记字段,见 LTAG,RTAG 305
Tail inequalities 尾部不等性 100,102—103
Tamaki Jeanne Keiko 玉置惠子 565
Tamari, Dow 塔马里,道 547
lattice 塔马里格 546,567
Tape, magnetic 磁带 132—133,435
paper 纸带 132—133,219,221
Tarjan, Robert Endre 塔简,罗伯特·恩德里 231,551
Taylor, Brook, formula with remainder 泰勒,布鲁克带余项的公式 113
series 泰勒级数 79,98
Temme, Nicolaas Maria 特姆,尼古拉斯·玛丽亚 64,118
Temporary storage 临时存储:在其它值占据寄存器时,用来短时间保存一个值的内存部分 183
Termial function 阶和函数 44,47
Terminal node of a tree 树的终端节点 293,302,334,375,567
Terminology 术语 Ⅷ,41,229,296,342,409
Ternary tree 三叉树 316,377
Tetrad tiling 平铺四位一体型 361—363
Tetrahedral arrays 四面体数组 286—287,291
$\TeX$ $\TeX$ 排版语言 Ⅸ,Ⅺ,184,192,580,630
Theory of algorithms 算法理论 6,8
Theory of automata 自动机理论 220,230,435—436
Thiele, Thorvald Nicolai 蒂利,索瓦尔德·尼古拉 99
Thorelli, Lars-Erik 托雷利,拉斯-埃里克 573,582
Thornton, Charles 桑顿,查尔斯 305,431—432
Thread an unthreaded tree 对一未穿线树穿线 316
Thread links 穿线链接 396
Threaded binary trees 穿线二叉树 305,314—315,432
compared to unthreaded 与未穿线加以比较 309
insertion into 插入到穿线二叉树 310,315

list head in 穿线二叉树中的表头 307,317
Threaded trees 穿线树 318—319,431
Three-address code 三地址代码 320,431
Tiling the plane 平铺平面 361
Time taken by a program 程序花费的时间,见 Execution time
Todd, John 托德,约翰 445
Todhunter, Isaac 托德亨特,艾萨克 174
Tonge, Frederic McLanahan, Jr. 小汤格,弗雷德里克·麦克拉纳汉 432
Top-down process 自顶向下方法 294,342
Top of stack 栈顶 229—230
Topological sorting 拓扑排序,拓扑分类 247—248,329,355,374
Torelli, Gabriele 托勒里.加布里埃尔 68,457
Toroidal tiling 环形平铺 362
Total ordering 全部次序,见 Linear ordering 256
Totient function $\varphi(n)$ 计数函数 $\varphi(n)$ 38,176
Trace routine 跟踪程字 184,203—205,220,282
Traffic signal 交通信号 155—156
Transfer instruction 转移指令:一个"跳转"指令
Transitive relation 传递关系,见 Ordering 104,247,335
Transposing a rectangular matrix 转置一个矩阵 175
Transposing blocks of data 转置数据块 176—177
Transpostions 转置:交换两个元素的排列 3,175,259,351
Traversal of binary trees 二叉树的遍历 303,430—431
inorder 二叉树的中根序遍历 304,306
postorder 二叉树的后根序遍历 314—315
preorder 二叉树的先根序遍历 314—315
Traversal of trees 树的遍历 319,431—432
prepostorder 树的先后根序遍历 539
Tree function $T(z)$ 树函数 372
Tree mappings 树映像 387
Trees 树 222,293—397
binary 二叉树,见 Binary trees
comparison of differnet types 不同类型树的比较 293—294,353
complete 完备树 378—379,381,534
construction of 树的构造 322—323,325,403—404
copying of 树的复制 312—313,316,329
definition of 树的定义 293,301,344,352
deletion from 从树中删除 339
Dewey notation for 树的杜威记号 297,301,361—362,432
diagrams of 树的图式 294—299,319,328,331,432
disjoint 树的拆散,见 Forest
embedding of 树的嵌入 330,364
enumeration of 树的枚举 364—376,384
equivalent 等价树 328
erasing of 树的抹去 316
free 自由树,见 Free trees
history 树的历史 382—383,431—432
infinite 无穷树 301,361
insertion into 插入到树 310,315
labeled, enumeration of 带标号树的枚举 366,383
linear ordering for 树的线性序 328
linked allocation for 树的链接分配 317,334—339
mathematical theory of 树的数学理论 343—384
ordered 有序树,见 Trees 293,253,366—367
oriented 有向树,见 Oriented trees
quadruply linked 四重链接树 339
representation of 树的表示 330—339,340—343,431—432
right-threaded 右穿线树 320,359
sequential allocation for 树的顺序分配 330—334,340—343,408
similar 相似树 328
t-ary t 叉树 317,374,378—379,381,563
ternary 三叉树 317,378
threaded 穿线树 318—319,431
traversal of 树的遍历 319,431—432
triply linked 三重链接 335,340—341,402—408
unordered 无序树,见 Oriented trees
unrooted 无根树,见 Free trees 344
Triangular matrix 三角矩阵 286,290
Triangulations of polygons 多边形的三角化 383,568—571
Tricomi, Francesco Giacomo Filippo 特里戈米,弗朗西斯科·贾科莫·菲利普 117,118
Tridiagonal matrix 三对角矩阵 292
Trigonometric functions 三角函数 40,219,441
Trilling, Laurent 特里林,劳伦特 433
Triple order for a binary tree 二叉树的三重序 538—539

Triply linked tree 三重链接的树 335, 340—341, 402—408
Trit 三进制数 135
Tritter, Alan Levi 特里特,艾伦·利瓦伊 547
Tucker, Alan Curtiss 塔克,艾伦·柯蒂斯 381
Turing, Alan Mathison 图灵,艾伦·马西森 16, 185, 219, 220, 431
  machines 图灵机 7, 220, 434
Tutte, William Thomas 塔特,威廉·托马斯 553
Twain, Mark (= Clemens, Samuel Langhorne) 吐温,马克(= 克莱门斯,塞缪尔·兰霍恩) 50
Twigg, David William 特威格,戴维·威廉 495
Two-line notation for permutations 排列的双线记号 158, 175
Two stacks 二栈 233, 238, 239—240
Two-way linkage 双向链接 265—266, 387
Typewriter 打字机 131—132, 221

Uhler, Horace Scudder 尤勒,霍勒斯·斯卡德 449
Ullman, Jefrey David 厄尔曼,杰弗里·戴维 532
UNDERFLOW 下溢标志 233, 234, 245, 254—255, 259
Uniform distribution 一致(均匀)分布:每个值都有同样可能性的一种概率分布 98, 240, 419
Unimodular matrices 单模矩阵 571
Uninitalized arrays 未初始化的数组 292
Union-find algorithm 惟一发现算法 335, 341
UNIVAC I computer UNIVAC I 计算机 146, 210, 221, 449
UNIVAC III computer UNIVAC III 计算机 120
UNIVAC 1107 computer UNIVAC 1107 计算机 120
UNIVAC SS80 computer UNIVAC SS80 计算机 120
UNIX operating system UNIX 操作系统 189
Unrooted trees 无根树,见 Free trees 344
Unusual correspondence between permutations 排列间的非常对应 171—172
Updates to memory, synchronous 存储器的同步更新 284
Uspensky, Vladimir Andreevich (Успенский, Владимир Андреевич) 乌斯片斯基,弗拉基米尔·安德烈维奇 435

van Aardenne-Ehrenfest, Tatyana 范阿登尼-埃伦费斯特,塔蒂亚纳 354, 358
van Ceulen, Ludolph 范休伦,卢道夫 566
van der Waerden, Bartel Leendert 范德瓦尔登,伯特尔·利恩德特 363—364, 557
van Leeuwen, Jan 范纽文,简 566
van Wijngaarden, Adriaan 范温加登,艾德里安 433
Vandermonde, Alexandre Théophile 范德蒙德,亚历山大·西奥菲尔 55, 67
  matrix 范德蒙德矩阵 33—34, 444
Vardi, Ilan 瓦迪,艾兰 474
Variable 变量:在程序执行中可以具有不同值的量 3, 225
  link or pointer 链接或指针变量 225
  node 节点变量 226
Variable-size nodes 可变大小的节点 409—410
Variance of a probability distribution 一个概率分布的方差 94
  deduced from the generating function 从生成函数推导的方差 96—99
Vauvenargues, Luc de Clapiers, Marquis de 沃维纳格斯,卢克·德·克拉皮尔斯,马奎斯·德 XIII
Vectors 向量,见 Linear lists
Velthuis, Frans Jozef 维尔休伊斯,弗兰斯·约瑟夫 630
Vertex in a graph 一图中的顶点 344, 351
  isolated 孤立顶点 353
Victorius of Aquitania 阿基坦的维克托里尤斯 489
Virtual machine 虚拟机器 191
Visit a node 访问一个节点 304
VLSI chips 超大规模集成电路芯片 535
von Ettingshausen, Andreas 冯·埃廷肖欣,安德烈亚斯 50
von Neumann, John (= Margittai Neumann János) 冯诺伊曼,约翰(= 马吉泰·纽曼·乔纳斯) 15, 219, 221, 429
von Segner, Johann Andreas 冯西格纳,约翰·安德烈亚斯 383, 508
von Staudt, Karl Georg Christian 冯斯托特,卡尔·乔治·克里斯琴 382

W-value in MIXAL MIXAL 中的 W 值 149—150
Wadler, Philip Lee 沃德勒,菲利普·李 574
Waerden, Bartel Leendert van der 瓦尔登,巴特尔·利恩德特·范·德 363—364, 557
Wait list 等候表,见 Agenda
Waite, William McCastline 韦特,威廉·麦卡斯特林

393,397,583
Wall, Hubert Stanley 沃尔,休伯特·斯坦利 450
Wallis, John 沃里斯,约翰 20,48
product for π π的沃里斯乘积 48,111
Wang, Hao 王浩 362
Waring, Edward 华林,爱德华 75,442
Warren, Don W. 沃伦,唐·W 341
Watanabe Masatoshi 渡边雅俊 630
Watson, Dan Caldwell 沃森,丹·考德威尔 238
Watson, George Neville 沃森,乔治·内维尔 477
lemma 沃森引理 119
Watson, Henry William 沃森,亨利·威廉 361
Weakest precondition 最弱先决条件 15
Weber, Helmut 韦伯,赫尔穆特 433
Webster, Noah, dictionary 韦伯斯特,诺亚词典 1,208
Wedderburn, Joseph Henry Maclagan 韦德伯恩·约瑟夫·亨利·麦克拉根 559
Wegbreit, Eliot Ben 韦格布雷特,埃利奥特·本 583
Wegner, Peter 韦格纳,彼得 219
Weierstrass[= Weierstraβ], Karl Theodor Wilhelm 维斯特拉斯,卡尔·西奥多·威廉 360
Weighted path length 带权的通路长度 378—381
Weiland, Richard Joel 韦兰,理查德·乔尔 577
Weizenbaum, Joseph 韦曾鲍姆,约瑟夫 390,431—433
Well-ordering 良序 18,317
Wheeler, David John 惠勒,戴维·约翰 219,220,429
Whinihan, Michael James 惠尼罕,迈克尔·詹姆斯 82
Whirlwind I computer 沃尔温德I计算机 220
Whitworth, William Allen 惠特沃思,威廉·艾伦 174
Wijngaarden, Andriaan van 温加登,艾德里安·范 433
Wilde, Oscar Fingal O'Flahertie Wills 王尔德,奥斯卡·芬格尔·奥弗拉蒂·威尔士 397
Wiles, Andrew John 威尔士,安德鲁·约翰 436
Wilf, Herbert Saul 威尔弗,赫伯特·索尔 61,62,90,454
Wilkes, Maurice Vincent 威尔克斯,莫里斯·文森特 219,220,429
Wilson, John, theorem 威尔逊,约翰定理 47
Wilson, Paul Robinson 威尔逊,保罗·罗宾逊 434
Windley, Peter F. 温德利,彼得·F 495
Windsor, House of 温莎王室 295
Winkler, Phyllis Astrid Benson 温克勒,菲利斯·阿斯特里德·本森 XI
Wirth, Niklaus Emil 沃思,尼克劳斯·埃米尔 182,433
Wise, David Stephen 怀斯,戴维·斯蒂芬 238,395,408,574
Wiseman, Neil Ernest 怀斯曼,内尔·欧内斯特 395
Wolman, Eric 沃尔曼,埃里克 435
Wolontis, Vidar Michael 沃伦提斯,维达尔·迈克尔 220
Woods Berners-Lee, Mary Lee 伍兹·伯内斯-李,玛丽·李 220
Woodward, Philip Mayne 伍德沃德,菲利普·梅恩 432
Word 字:计算机存储器的可寻址的单元 121—123
Word size, for MIX MIX的字的大小:可以存入五个字节中的不同值的个数
Wordsworth, William 沃德沃思,威廉 135
Worst-fit method of storage allocation 存储分配的"最坏适合"方法 425,577
Wortman, David Barkley 沃特曼,戴维·巴克利 220
Wrench, John William, Jr. 小伦奇,约翰·威廉 449,590
Wright, Edward Maitland 赖特,爱德华·梅特兰 461,492
Wright, Jesse Bowdle 赖特,杰西·鲍德尔 340
Writing 写:进行输出 207
Writing large programs 写大型程序 182—184
Wyman, Max 怀曼,马克斯 62
Wythoff (= Wijthoff), Willem Abraham 威索夫,威廉·亚伯拉罕 77

X-1 computer X-1计算机 221
X-register of MIX MIX的X寄存器 121
XDS 920 computer XDS 920计算机 120
XOR (exclusive or) XOR指令(异或) 427

Yang Hui 杨辉 50
Yao, Andrew Chi-Chih 姚期智 512
Yngve, Victor Huse 英韦,维克多·休兹 432
Yo-yo list 唷唷表 229
Yoder, Michael Franz 约德,迈克尔·弗朗兹 448

Young, David Monaghan, Jr. 小扬,戴维·莫纳罕 554

z 用来表示复数 18,104
Zabell, Sandy Lew 泽贝尔,桑迪·卢 460
Zave, Derek Alan 泽夫,德里克·艾伦 88,583
Zeckendorf, Edouard 泽肯多夫,埃杜瓦德 464
Zeilberger, Doron 泽尔伯格,多龙 61
Zemanek(= Zemánek), Heinz 泽曼尼克,海因兹 1
Zeitschrift für Math. und Physik 《数学和物理杂志》
Zeitschrift für Physik 《物理学杂志》
Zeta function $\zeta(s,x)$ 仄塔函数 40,73
Zhang, Linbo 张林波 630
Zimmerman, Seth 齐默尔曼,赛思 384
Zorn, Max, lemma 佐恩,马克斯引理 518
Zuse, Konrad 佐斯,康拉德 429

We must not . . . think that computation,
that is ratiocination,
has place only in numbers.
我们不应……认为计算,
或者推论,
只会在数值中出现。
——THOMAS HOBBES, *Elementary Philosophy* (1656)

本书的写作是在安装了 Computer Modern 字体的一台 Sun SPARCstation 上完成的,使用了 TEX 和 METAFONT 软件[作者的*Computers & Typesetting* (Reading, Mass.: Addison-Wesley, 1986), Volumes A ~ E 中对此作了介绍]。插图用 John Hobby 的 METAPOST 系统制作。索引中的一些名词用到了其它一些字体,它们是由 Yannis Haralambous (Greek, Hebrew, Arabic), Olga G. Lapko (Cyrillic), Frans J. Velthuis (Devanagari), Masatoshi Watanabe (Japanese), 和 Linbo Zhang (Chinese)开发的。——原注

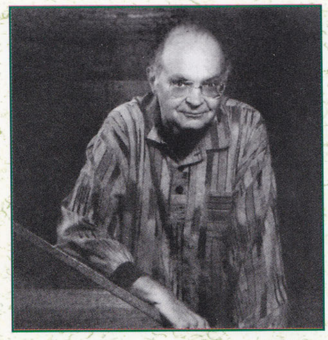
内 容 简 介

本书是国内外业界广泛关注的7卷本《计算机程序设计艺术》第1卷的最新版，以基本的程序设计概念和技术开始，然后专注于信息结构——信息在计算机内部的表示、数据元素之间的结构关系以及如何有效地处理它们，给出了对于模拟、数值方法、符号计算、软件和系统设计的初等应用。书中附有大量习题和答案，标明了难易程度及数学概念的使用。

新版本增加了几十项简单且重要的算法和技术，并对有关数学预备知识作了大量修改以适应现时研究的趋势。

本书可供从事计算机科学、信息科学、计算数学、计算技术、科技史诸方面的工作人员参考、研究和借鉴，也是相关专业高等院校的理想教材和教学参考书。

本次印刷已根据 http://www-cs-faculty.stanford.edu/~knuth/taocp.html/ 上的最新勘误表（截止到2007年5月2日）对重要疏误做了校正。



Donald E. Knuth（唐纳德·E·克努特，中文名高德纳）是算法和程序设计技术的先驱者，是计算机排版系统T_EX和METAFONT的发明者，他因这些成就和大量创造性的影响深远的著作（19部书和160篇论文）而誉满全球。作为斯坦福大学计算机程序设计艺术的荣誉退休教授，他当前正全神贯注于完成其关于计算机科学的史诗性的七卷集。这一伟大工程在1962年他还是加利福尼亚理工学院的研究生时就开始了。Knuth教授获得了许多奖项和荣誉，包括美国计算机协会图灵奖（ACM Turing Award），美国前总统卡特授予的科学金奖(Medal of Science)，美国数学学会斯蒂尔奖（AMS Steele Prize），以及由于发明先进技术而于1996年11月荣获的极受尊重的京都奖(Kyoto Prize)。现与其妻Jill生活于斯坦福校园内。

欲了解这位杰出科学家和作家的更多信息,请访问
www.aw.com/cseng/authors/knuth

欲了解本书和其它各卷的更多信息,请访问
www-cs-faculty.stanford.edu/~knuth

欲了解中译本的更多信息,请访问
www.ndip.com.cn/computer/taocp

经典计算机
科学著作
最新修订版

计算机科学/程序设计

它本来是作为参考书撰写的，但有人却发现每一卷都可以饶有兴致地从头读到尾。一位中国的程序员甚至把他的阅读经历比做读诗。

如果你认为你确实是一个好的程序员，读一读Knuth的《计算机程序设计艺术》吧，要是你真把它读通了，你就可以给我递简历了。 ——Bill Gates

这一鸿篇巨制被广泛誉为对经典计算机科学的权威描述，头三卷在几十年来一直是学生、研究人员和业内人士的无价财富。

一部所有基础算法的宝典，今天的许多软件开发者关于计算机程序设计的绝大多数知识都是从该宝典中获得的。 ——*Byte*, September 1995

不计其数的读者深受Knuth著作的影响。科学家们惊讶于他的分析的精美与雅致，而普通程序员每天都从其提供的“菜谱”中获得问题解决方案。书的恢宏、透彻、精确与幽默赢得了所有人的尊敬。

我无法描述它们在我学习和娱乐中伴我度过的愉悦时光。我在车里，在餐馆里，在家里……甚至在我儿子棒球小联赛的间隙都忘不了带上它们，一有空就捧出来阅读。 ——Charles Long

不管你基础如何，倘若你想认真地编写任何计算机程序，你都有理由把这套书的任何一卷抱回家，以便在你学习和工作的时候随时翻阅。

要是有一个问题难到要把Knuth的著作请下书架，那就太令人愉悦了。我发现，人们只要翻一翻它，就会对计算机产生极其有用的令人“恐慌”的影响。 ——Jonathan Laventhol

为反映该领域的最新发展，Knuth二十多年来第一次将三卷书全部做了修订。他的修订主要集中在上一版以来趋于一致的知识，已经解决的问题，以及有所变化的问题。为保持本书的权威性，关于该领域先驱工作的历史信息都做了更新。为维护作者苦心孤诣追求至善至美的盛誉，新的版本将敏锐和苛刻的读者发现的少量技术错误都做了更正。增加了上百的新习题，对您也是新的挑战。

责任编辑：辛再甫 zfxin@ndip. com.cn

计算机程序设计艺术

第1卷 基本算法

(第3版)

http://www.ndip.cn
ISBN 978-7-118-02799-0
9 787118 027990 >
国防工业出版社
National Defense Industry Press
http://www.ndip.cn
ISBN 978-7-118-02799-0
定价：98.00元
Addison-Wesley
Addison
Wesley